STUDENTS’ UNDERSTANDING OF RECURSION
WHEN DO THEY USE IT AND WHY?

By
Adi Levy Conlogue
B.S. University of Maine, 2009

A THESIS
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Teaching

The Graduate School
The University of Maine
May 2013

Advisory Committee:
Natasha M. Speer, Assistant Professor of Mathematics Education, Co-Advisor
Roy M. Turner, Associate Professor of Computer Science, Co-Advisor
MacKenzie Stetzer, Assistant Professor of Physics
THESIS ACCEPTANCE STATEMENT

On behalf of the Graduate Committee for Adi Levy Conlogue, I affirm that this manuscript is the final and accepted thesis. Signatures of all committee members are on file with the Graduate School at the University of Maine, 42 Stodder Hall, Orono Maine.

____________________________________________________________
Natasha M. Speer, PhD

____________________________________________________________
Roy M. Turner, PhD
LIBRARY RIGHTS STATEMENT

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at The University of Maine, I agree that the Library shall make it freely available for inspection. I further agree that permission for "fair use" copying of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature:

Date:
STUDENTS’ UNDERSTANDING OF RECURSION

WHEN DO THEY USE IT AND WHY?

By Adi Levy Conlogue

Thesis Advisors: Dr. Natasha M. Speer
Dr. Roy M. Turner

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Teaching
May 2013

Recursion is the process of repeating items in a self-similar way. Recursion is a key concept in the Computer Science field and is used in programming. It is a powerful tool for solving programming tasks and has features that sometimes make it a superior choice over other approaches. Students learn recursion during their first programming course and other courses throughout the curriculum. Research has shown that recursion is challenging and findings reveal students’ difficulties in understanding and applying it to solve problems. But very little is known about when students choose to use recursion to solve programming tasks and why they do or do not choose to use it. Investigating students’ thinking about the use of recursion is the focus of this study. Participants included 17 undergraduates and three graduate students. Task-based clinical interviews were the sources of data. Findings indicate that students do not write functions that use recursion to solve programming tasks even though they are actually able to successfully use recursion when asked to do so. The analysis sheds some light on various reasons why students who are capable of using recursion choose not to use it. Implications for teaching, limitations of this study, and future research directions will be discussed.
DEDICATION

For Grandpa, I hope I made you proud.

For Elise, without you I would not have pursue an education degree.

For Sivan, Inbar, Amit, Netta and Kathrina, you are the next generation and I hope you will take advantage of the opportunities given to you.

Last, but not least, to my husband Ryan, you are my support, my rock and the love of my life.
ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Natasha Speer. Thank you for your help and your support through the process of writing this thesis. This thesis would not have been possible without her guidance, her kind words and her supporting me when I doubted myself. I truly enjoyed our meetings and our conversations. Thank you for helping me develop my research skills, writing skills and teaching skills that are required for this field. I am proud to say I learned from you!

I would also like to thank my co-advisor Dr. Roy Turner. Over the past eight years he has been someone I’ve always know I could count on. I enjoyed spending Christmas, Thanksgivings and “just because” days with the Turners. Thank you for being there for me when I broke down and cried, thank you for all your support.  Roy, you are a true inspiration to me.

In addition I would like to thank my family. It has not been an easy journey. Being 8500 km (or 5281.65 miles) away from home took a big toll on me. I miss my family every day. Thank you for your support, for believing in me, for the endless Skype calls throughout the night, for making me laugh when I wanted to cry and for just being there for me. Mom, thank you for pushing me to be better then I thought I could be. You helped me become the person I am today. Dad, you are the best! You chose me to be your daughter. Thank you for the night talks and the support and the love you gave me every step of the way. Grandma, I love you. Talking to you every morning has pushed me to finish strong and I am happy to say that our friendship is something very special to me!

I have the best family and I feel like the luckiest girl in the world. Thank you for the unconditional support and love. Without you it would not have been possible.
# TABLE OF CONTENTS

DEDICATION ......................................................................................................................... iv

ACKNOWLEDGEMENTS ......................................................................................................... v

CHAPTER 1 - INTRODUCTION AND OVERVIEW ................................................................. 1

1.1 Introduction and rationale ............................................................................................. 1

1.2 Overview of study ........................................................................................................... 7

1.3 Goal of study and research questions ............................................................................ 9

1.4 Abstract of first findings chapter: students solving programming tasks ...................... 9

1.5 Abstract of second findings chapter: students Solve Programming Tasks

Using Recursion .................................................................................................................... 10

1.6 Abstract of third findings chapter: students’ choosing recursive Solutions ............... 10

CHAPTER 2 - RESEARCH ON LEARNING AND TEACHING RECURSION .............. 12

2.1 Theoretical perspective ............................................................................................... 13

2.2 Definition of recursion and the ideal knower ............................................................... 14

2.3 Why knowing about students’ thinking about recursion matters .................................. 16

2.4 Recursion is important and difficult ............................................................................. 18

2.5 Teaching recursion is challenging ................................................................................ 20

2.6 What do novice and non-novice programmers know about

recursion and what are their difficulties? ........................................................................... 23

2.7 When should recursion be taught? ................................................................................ 24

2.8 When Do Students Use Recursion And Why? ............................................................... 25

2.9 Summary ....................................................................................................................... 26
CHAPTER 3 - METHODOLOGY ........................................................................................................... 28

3.1 Methods of data collection ........................................................................................................... 28

3.1.1 Subjects of research ................................................................................................................... 28

3.1.2 Interviews .................................................................................................................................. 29

3.1.2.1 Background questions ........................................................................................................... 29

3.1.2.2 Part 1 .................................................................................................................................... 29

3.1.2.3 Part 2 .................................................................................................................................... 34

3.1.2.4 Part 3 .................................................................................................................................... 36

3.1.2.5 Part 4 .................................................................................................................................... 37

3.1.2.6 Part 5 .................................................................................................................................... 38

3.2 Methods of data analysis ............................................................................................................... 38

3.2.1 Horizontal analysis .................................................................................................................... 39

3.2.1.1 Development of analysis for part one of the interview ......................................................... 40

3.2.1.2 Application of analysis to part one of the interview ............................................................... 45

3.2.1.3 Development of analysis of part two of the interview ......................................................... 47

3.2.1.4 Application of analysis to part two of the interview ............................................................... 48

3.2.1.5 Development of analysis for part three of the interview ..................................................... 49

3.2.1.6 Application of analysis to part three of the interview ............................................................. 50

3.2.1.7 Development of analysis for part four of the interview ....................................................... 52

3.2.2 Vertical analysis .......................................................................................................................... 53
CHAPTER 4 - STUDENTS SOLVING PROGRAMMING TASKS

4.1 Students do not choose to solve programming tasks using recursion:

results from part one of interview .......................................................... 56

4.1.1 Power Task findings. ................................................................. 57

  4.1.1.1 Iteration: For loop and while loop responses .......................... 58
  4.1.1.2 Correct responses ............................................................... 59
  4.1.1.3 Iteration: incorrect responses ............................................ 61

4.1.2 Summation of digits findings ..................................................... 62

  4.1.2.1 Iteration: For loop, while loop, and responses ...................... 63
  4.1.2.2 Iteration: correct and incorrect responses ............................ 64

4.1.3 List Search Task findings ......................................................... 70

  4.1.3.1 Iteration: For loop and while loop responses ....................... 72
  4.1.3.2 Correct and incorrect responses ........................................ 73

4.1.4 Summary of part one results ................................................... 75

CHAPTER 5 – STUDENTS SOLVE PROGRAMMING TASKS USING RECURSION

5.1 Students can write recursive solutions when prompted:

Results from part three of the interview ................................................. 77

5.1.1 Power Task findings. ................................................................. 78

  5.1.1.1 Correct responses ............................................................. 79
  5.1.1.2 Incorrect responses ......................................................... 79
  5.1.1.3 Did not solve the task responses ....................................... 82
CHAPTER 6 - STUDENTS’ CHOOSING RECURSIVE SOLUTION ................. 107

6.1 Students prefer the recursive solution when given a choice
    between two different solutions: Results of part two of the interview .......... 107

6.1.1 Factorial task findings ........................................................................ 108

6.1.2 Response categories ............................................................................. 109

6.1.2.1 Stack overflow category. ................................................................. 110

6.1.2.2 Properties of recursion category. ..................................................... 111

6.1.2.3 Lack of confidence in a solution category. ....................................... 112

6.1.2.4 “other” category ............................................................................. 113

6.1.3 Binary tree task findings. ..................................................................... 113

6.1.4 Response categories ............................................................................. 114

6.1.4.1 Structure resemblance category. ..................................................... 115

6.2 Participants chose the recursive solution when presented
    with two different solutions. ....................................................................... 116

6.2.1 Common recursive examples: Factorial and Binary Tree....................... 118

6.2.2 Reasons for choosing iteration: Class related. ..................................... 119

6.2.1 Recursive solution and base case. ....................................................... 120

6.3 Participants modeled the tasks using different models:
    Results of part four of the interview ......................................................... 123

6.4 Discussion of Participants modeled the tasks .......................................... 126
CHAPTER 7 – CONCLUSIONS, IMPLICATIONS & SUGGESTIONS

FOR FURTHER RESEARCH

7.1 Students do not chose to solve algorithmic programming tasks using recursion

7.2 Students can write recursive solutions when prompted

7.2.1 Reasons students solved tasks using iteration

7.3 Students prefer the recursive solution when given a choice between two different solutions

7.3.1 Reasons students chose a recursive solution

7.4 Participants modeled the tasks using different models

7.5 Limitation of study and further research

REFERENCES

BIOGRAPHY OF THE AUTHOR
LIST OF FIGURES

Figure 1.1. Recursive example: expansion of Russian Nesting doll.............................. 2

Figure 1.2. Recursive example: first recursive step of
counting Russian Nesting dolls................................................................. 3

Figure 1.3. Recursive example: second recursive step of
counting Russian Nesting dolls................................................................. 3

Figure 2.1 example of a recursive programming function........................................ 15

Figure 3.1. Power Task ............................................................................................ 30

Figure 3.2. Summation of Digit Task ........................................................................ 30

Figure 3.3. The List Search Task ................................................................................ 30

Figure 3.4. Factorial task .......................................................................................... 35

Figure 3.5. Binary tree task....................................................................................... 35

Figure 3.6. Recursive factorial task .......................................................................... 36

Figure 3.7. Recursive summation of digits task.......................................................... 36

Figure 3.8. Recursive List Search Task ...................................................................... 37

Figure 3.9. Visual representation of patterns within data for the first three tasks .......... 41

Figure 3.10. A correct for loop solution to the Power Task ........................................ 42

Figure 3.11. Flow chart of an iterative for loop solution to the Power Task ................ 44

Figure 3.12 A correct while loop example to the Power Task.................................... 45

xii
Figure 4.11. Incorrect iterative solution to the Summation of digit task using for loop with division operations .......................................................... 70

Figure 4.12. List Search Task results .................................................................................................................. 72

Figure 4.13. Correct iterative solution to the List Search Task, using a for loop structure .................................................. 74

Figure 5.1. Power Task results ............................................................................................................................ 79

Figure 5.2. An incorrect recursive solution to the Power Task .............................................................................. 80

Figure 5.3. An incorrect recursive solution to the Power Task .............................................................................. 81

Figure 5.4. An incorrect recursive solution to the Power Task .............................................................................. 82

Figure 5.5. A correct iterative solution to the Power Task using a for loop ............................................................ 83

Figure 5.6. Summation of Digits Task results ....................................................................................................... 86

Figure 5.7. Incorrect response to Summation of Digits Task .................................................................................. 87

Figure 5.8. Incorrect response to Summation of Digits Task .................................................................................. 88

Figure 5.9. List Search Task results .................................................................................................................... 90

Figure 5.10. Incorrect response to the List Search Task ......................................................................................... 92

Figure 6.1 Example of a binary tree as a recursive structure ................................................................................. 116

Figure 6.2 Example of a list as a recursive structure ............................................................................................. 125
LIST OF TABLES

Table 3.1 Example of tracing table for calculating $4^2$ ................................................. 47

Table 4.1. Results of the first three tasks ............................................................................. 75

Table 5.1 Accuracies of participants for each task using recursion................................. 95

Table 5.2 Frequency of participants’ solution accuracies using recursion for each task ................................................................. 95

Table 5.3 Response categories to why solving the tasks iteratively ................................. 96

Table 6.1. Response categories why students chose one solution over another when considering the factorial task ................................................................. 109

Table 6.2. Response categories why students chose the recursive solution when considering the binary tree task ................................................................. 114
CHAPTER 1 – INTRODUCTION AND OVERVIEW

1.1 Introduction and rationale

Technology is an important part of our daily lives. The technology field relies on engineers, mathematicians and scientists to make profound and important changes that will improve the field and improve our lives. We need Computer Scientists to be able to solve tasks and use programming to create various products that will be useful for us. The demand for computing and programming jobs is rising. In order to produce great programmers and help fill the available jobs, we need to improve the enrollment, retention and graduation rates of students in undergraduate programs in the Computer Science field.

The decline in enrollment rates for computer science degree programs is a nationwide phenomenon (Beaubouef & Mason, 2005; Cohoon, Wu, & Luo, 2008; Powell, 2008; Sloan & Troy, 2008). Research supports the claim that even those who choose to enroll in a computer science program feel that it is a tough major, with “hard” concepts to grasp (Rao & Mitra, 2008). Many students do not complete the first programming course, and of those who do, many drop the Computer Science major (Beaubouef & Mason, 2005; Rao & Mitra, 2008). The first course in the computer science major is typically a programming course. If the field is losing students after their initial exposure to programming, one strategy for improving retention and graduation rates is to find ways to help more students succeed in that initial programming course. One component of introductory programming courses that is known to be especially challenging is recursion.
Recursion is a fundamental and key concept in computer science. Recursion is defined as the process of repeating items in a self-similar way. An example of recursion is counting the number of dolls inside a Russian Nesting Doll. Russian Nesting dolls are a set of wooden dolls of decreasing size placed one inside the other. This is considered to be a recursive structure because it involves repeating items in a self-similar way. Figure 1.1 shows a traditional Russian Nesting Doll expanded from the largest doll to the smallest doll.

If one desires to count the number of dolls in a Russian Nesting doll, one can use recursion. As all the smaller dolls are inside the largest doll, one can open the first doll and reveal the second largest doll. The total number of dolls is the first doll, plus the rest of the dolls (that are inside the second largest doll). Figure 1.2 represents this instance.
This step can be repeated as one could count another doll (the second one from the previous step), plus the rest of the dolls (that are in the third largest doll). Figure 1.3 represents this instance. One could open the doll and find another smaller doll inside of it. This could be repeated until the smallest doll is revealed. To find the total number of dolls, the smallest doll will return 1 to your count of the number of dolls and this is done for each doll until the largest doll is reached, at which point you will have the correct count of the number of dolls. This is considered to be a recursion metaphor because each smaller doll represents a smaller version of the same “problem” of the larger doll.
The term recursion has multiple meanings across different disciplines. The most common applications of recursion are in the fields of mathematics and computer science. In computer science, it is a method in which the solution to a problem depends on solutions to smaller instances of the same problem. In mathematics, recursion is a method of defining a mathematical function based on previously defined terms of the same function.

Many different problems can be defined in a recursive manner. For example, when calculating a factorial, one can use recursion. In order to calculate 3 factorial, denoted as 3!, one can make use of the answer for 2!, since 3! is equal to 3 \cdot 2!. In a more general form, when one tries to evaluate n!, one can make use of the definition of (n-1)!. Hence, the definition of factorial can be stated as n! = n \cdot (n-1)!. The calculation will continue to occur with a smaller input until it gets to zero factorial (0!), which is equal to 1. In addition to a recursive definition, factorial can be defined iteratively. Iteration is defined as the act of repeating a process with the aim of approaching a desired result. Factorial can be defined as n \cdot (n-1) \cdot (n-2)\ldots \cdot 1, which is considered to be an iterative approach because one is repeating the multiplication operation with decrementing numbers to calculate the result; you are repeating the same process with the aim of calculating factorial.

Each repetition of the process is also called ”iteration,” and the results of any iteration are used as the starting point for the next iteration. Iteration in mathematics may refer to the process of applying a function repeatedly, for example, counting the numbers from 1 to 100, incrementing each time by 1. Iteration in computer science is the repetition of a block of statements within a computer program.
Every recursive solution can be implemented using iteration and vice versa. One might wish to use a recursive solution instead of an iterative solution because recursive solutions are typically shorter to write, more elegant, and simpler to read. In contrast, iterative solutions can be longer and more complicated.

Recursion has been identified as an important topic by the Association for Computing Machinery, also called ACM. The Association for Computing Machinery (ACM) is widely recognized as the premier membership organization for computing professionals, delivering resources that advance computing as a science and a profession, enabling professional development and promoting policies and research that benefit society (ACM, 2001).

The ACM curriculum suggests four academic classroom hours dedicated to recursion during the first programming course. Recursive structures also appear in upper-level courses throughout the degree, and recursion is necessary as a programming tool and problem-solving skill. Having a strong understanding of recursion and its uses is important in order to continue study within the Computer Science field and to seek a career in science, technology, engineering and mathematics (STEM) fields. Therefore, it is vital that we find ways to increase students’ understanding of the concept of the recursion and their abilities to apply it when appropriate.

Concerns about enrollment and retention rates, in addition to students’ understanding of programming and desires to improve teaching of the introductory computer courses, have sparked research over the past decades. Researchers have found that recursion is a hard concept to learn (Anderson, Prirroli & Farrell, 1988; Dicheva & Close, 1996; Gotschi, Sanders & Galpin, 2003; Haberman & Averbuch, 2002; Kahney, 1989; Kessler
Anderson (1989). Ginat (2004) concluded that students do not use recursion as a problem-solving tool. Instead of using recursion, students try to solve the task using various non-recursive approaches, some of which are unsuccessful.

Very little is known about when students choose to use recursion to solve programming tasks and why they do or do not choose to use it. The primary goal of this study is to identify when students use recursion as an approach to solving various programming tasks. Recursion is an important topic in the curriculum and it is important for students to understand how to use it, but it is also important for students to know when to use it. Students who have a hard time understanding the topic of recursion and implementing and constructing recursive functions might be at a disadvantage at the university level since recursion might be needed in order to solve certain programming tasks, or to understand particular structures and algorithms. Therefore, it is important to investigate and understand what makes recursion difficult as well as when and why students use recursion as a problem solving technique. Such information could perhaps shed some light on how we teach recursion. Improving how we teach recursion can result in additional students using recursion successfully as a tool to solve programming tasks. This could lead to greater success of students in subsequent computer science courses.

Research on students’ understanding of recursion in the computer science education field has been growing. Researchers have shown that students struggle with different elements of recursion, such as the base case and the passive flow of the recursive function. These terms are defined in chapter two. In addition, students struggle with constructing recursive functions and have difficulties evaluating recursive algorithms (Haberman & Aviv, 2002). However, very little is known about how these
difficulties relate to students’ use of recursion to solve programming tasks and why they do or do not choose to use it.

Some populations of students are underrepresented in the field of Computer Science, particularly at the highest levels, such as doctoral degrees (Cohoon et al., 2008; Palma, 2001; Rodger & Walker, 1996; Wilson, 2002). Understanding how students think about concepts such as recursion can help educators with the difficulties of teaching the abstract concept. Beyond recruiting students from underrepresented groups, this knowledge can improve the teaching of computer science to all students, thereby improving retention and graduation rates.

Answering when and why students use recursion can help instructors and educators help students to understand recursion during their undergraduate studies. One useful application of this knowledge is the ability to plan for instruction and to help students overcome difficulties. If a there is a certain reason why most students avoid using recursion, educators will be able to anticipate this and improve their strategies in order to address these difficulties. For example, if a student does not chose to use recursion because of the fear of a stack overflow, a teacher might be able to show several examples where recursion is useful and stack overflow is not occurring.

1.2 Overview of study

The goal of this study is to develop descriptions of students’ understanding of recursion while solving programming tasks and to describe instances when students did or did not chose to use recursion as an approach to solve programming tasks. Participants were asked to give reasons for using or not using recursion. These reasons are presented
and discussed in this thesis. Participants included 17 students from different disciplines at a large public university in the northeastern U.S. They participated in hour-long clinical interviews (Hunting, 1997). Clinical interview transcripts and students’ written and/or verbal response to programming tasks were analyzed. The interview was divided into four different parts. During the first part, participants were asked to solve three different programming tasks. The first task, the power task, asked participants to write a function to calculate $x^n$. The second task, the summation of digits task, asked participants to sum the digit of a positive integer. The third task, the search list task, asked participants to write a function that will return true if a given item is within a list, otherwise return false.

Findings from the study indicate that students do not use recursion as their primary chosen approach when solving programming tasks. However, when presented with two correct solutions, one iterative and one recursion, students choose (and prefer) the recursive solution as an appropriate solution to solve programming tasks. In addition, findings offer various reasons why students do not use recursion and prefer iteration.

In the rest of this chapter, the researchable questions are presented. Chapter two contains the theoretical perspective that framed this study, in addition to a review of the relevant literature. Chapter three contains the research design and interview instrument, the methods of data collection and data analysis. Chapter four, five and six contain findings and discussion of the answers to the researchable questions. Finally, chapter seven contains summary of findings, conclusions, a discussion of the limitations of the present research, and suggestions for future research
1.3 Goal of study and research questions

The following are the particular researchable questions that frame this study:

• Do students choose to solve algorithmic tasks using recursion without prompting?
  – If so, why do they choose to use recursion?
  – If not, why do they choose to use iteration?

• If students use recursion, do they define the problem as a set of sub-problems?
  – Do they define the objects of the problem as recursive structure?

Shavelson & Towne (2002) categorized research questions into three different types: description questions (What is happening?), cause questions (Is there a systematic effect?), and process or mechanism questions (Why or how is it happening?). According to these categories my research questions are defined as process/mechanism questions because they are aimed to discover why students chose to use recursion as a tool to solve programming tasks.

1.4 Abstract of first findings chapter: students solving programming tasks

Findings from part one of the interview data show that student do not choose to solve algorithmic programming tasks using recursion. These findings suggest that students can correctly and successfully solve these programming tasks using iteration, which is their preferred approach. There can be many possible explanations for student choosing to use iteration as their primary approach to solve programming tasks. Participants were asked for reasons for choosing iteration, and these findings are discussed in Chapter 5.
1.5 Abstract of second findings chapter: students Solve Programming Tasks Using Recursion

During part three of the interview, participants were asked to solve the same three programming tasks but this time they were prompted to use the other approach. For example, if a participant initially solved the power task using iteration, in part three he or she was asked to solve it using iteration. Findings from part one show that participants choose to use iteration as their chosen approach. Therefore, part three asked participants to solve the power task, summation of digits task and search list task using recursion. Findings from part three of the interview show that, despite their preference for iteration, students can successfully solve algorithmic programming tasks using recursion. After solving the three tasks using recursion, participants were asked to state a reason why recursion was not their initial approach. Findings show that students said that recursion can cause stack overflow, and therefore they choose to solve tasks using iteration. In addition, findings suggest that students believe iteration is their chosen approach because they were taught iteration prior to recursion. Additional reasons are also discussed in this chapter.

1.6 Abstract of third findings chapter: students’ choosing recursive Solutions

Part two of the interview asked participants to choose between a correct iterative solution and a correct recursive solution to solve two different tasks. The first task in this part of the interview, the factorial task, asked the participant to choose a solution that will calculate factorial to a given integer n. The second task, the binary tree task, asked the participants to choose a solution that will count the number of nodes in a binary tree.
Findings from part two show that even though participants do not choose to solve tasks using recursion, they would pick the recursive solution over the iterative solution. These findings suggest that students might be more apt to choose a recursion solution when it is present, but they do not choose to write a recursive solution without prompting.

Participants were asked to specify reasons as to why they would choose a recursion solution when two solutions are presented. Findings show that participants chose the iterative solution because they believed that recursion is more elegant, simple and readable, whereas the iterative solution is long and complicated. Additional reasons are also discussed in this chapter.
Research has shown that recursion is hard concept to learn (Anderson, Prirolli & Farrell, 1988; Dicheva & Close, 1996; Gotschi, Sanders & Galpin, 2003; Haberman & Averbuch, 2002; Kahney, 1989; Kessler & Anderson 1989). Many educators describe recursion as an abstract concept difficult to teach (Bruce, Danyluk & Murtagh 2005; Dann, Cooper & Pausch 2001; Ford 1982; Turbak, Royden, Stephan & Herbst 1999; Velazquez-Itrubide 2000). Leron (1988) hypothesized that what makes recursion inherently hard to understand is the complexity gap between the recursive calls and the processes they generate. Even though the recursive call is visually present within the recursive function, the passive flow and building up the result, is a non-visual feature of recursion. Recursion is a powerful tool to solve many programming tasks in computer science. Usually recursion is taught during the first programming course where students are prompted to use recursion in order to practice this approach. Later in the curriculum, when recursive structures and algorithms are taught, students are asked to use recursion in order to solve programming tasks. Ginat (2004) noted that students are explicitly asked to employ recursion in the chapter in which it is introduced and in later courses students use recursion when practicing particular topics that involve recursion (e.g., recursive algorithm or recursive structures). However, little is known about whether (and how) students identify the programming problems for which recursion should be used. Some programming tasks are easily solved using recursion. It would be ideal for students to identify these tasks when recursion should be implemented and successfully construct recursive solutions. However, Ginat (2004) found that when given programming
problems that can be solved using recursion, the majority of students did not use the
recursive approach. Instead, they attempted various forward reasoning ways, such as
iteration, some of which were unsuccessful at solving the given task. Even though
Ginat’s (2004) study and the current study share some similarities there are some
differences between the two. These similarities and differences are described in this
chapter.

It is vital that students know how to use recursion when solving programming
tasks, but it is also important for students to know when to use it. However, very little is
known about this issue. The aim of this study is to investigate when students use
recursion when solving programming tasks and their reasons for why they choose to do
so.

2.1 Theoretical perspective

Cognitivism is a theory that attempts to answer how and why people learn by
attributing the process to cognitive activity (Byrnes, 2001). This study was conducted
within a cognitivist framework. Because the aim of this research is to investigate when
students choose to use recursion to solve algorithmic programming tasks and why, a
cognitive theoretical perspective was appropriate for this research. Because little is
known about when students use recursion and why, there are no similar research studies
that used this theoretical perspective; however a cognitive theoretical perspective has
been used in other work on similar topics in the field of computer science education
research (Ginat & Shifroni, 1999; Haberman & Aviv, 2002; Levy & Lapidot, 2000,
2002).
2.2 Definition of recursion and the ideal knower

Recursion is defined as the process of repeating items in a self-similar way. In mathematics, recursion is a method of defining a mathematical function based on previously defined terms of the same function. Many mathematical axioms are based upon recursive rules. In computer science, recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. Many computer science algorithms and computer science structures are recursive algorithms and recursive structures respectively.

When programming a recursive function, ideally a student should include three different elements in order to construct a successful recursive function. The three different elements are: active flow, which is the recursive call, base case, which is the terminal condition of the function and passive flow, which is the result of the function. The first element is called the recursive call. The recursive call is a statement where the function called itself with smaller input. Figure 2.1 presents a recursive function that calculates n factorial. For example, the last return statement in Figure 2.1 presents a recursive call where the function calls itself with a smaller input (n is decremented by 1). The second element of the recursive function is the base case. The base case is the also the halting case of a function. The base case can be defined as the stopping condition, where the function will stop executing the recursive call and a value will be calculated. The if statement in Figure 2.1 represents the function’s stopping condition, where n is equal to zero. The third feature is also called the passive flow where a value is passed back to previous invocations and a solution is built. The return statement in Figure 2.1
represents the passive flow where the result flows from the terminated function to
previous function call in order to build the final solution.

![Factorial (n)

if (n = 0) then
    Return 1
Else
    Return ( n \cdot \text{Factorial} (n-1))

Figure 2.1 example of a recursive programming function

Recursion can be finite or infinite. A finite recursive definition includes within it a
stopping condition or a base case, which is the level beyond which it cannot be reduced
further. For instance, using the factorial example above, zero factorial is acting as the
base case. When the recursive calculation gets to zero factorial, the result is 1, and
recursion stops. The function will halt, and will not try to calculate \((-1)!\)

An infinite recursion involves a recursive process without a stopping condition.
For example, summing all the integers would be an infinite process. Because the domain
of all integers is infinite, a base case or a stopping condition cannot be identified and
written; therefore the calculation will occur infinitely many times.

It is not very useful for a computer scientist or a programmer to write an infinite
recursive function due to the fact that it will not yield a correct output, or indeed, any
output at all. Infinite recursion will continue to execute until the user manually breaks
from or stops the function or a stack overflow occurs.

Stack overflow is a situation that occurs when too much memory is used on the
call stack. The call stack is an area of memory in which data associated with calling a
function is placed. For recursive functions, each recursive call will cause information to be stored on the stack. The call stack contains a limited amount of memory, often determined at the start of the program. The size of the call stack depends on many factors, including the programming language, machine architecture, multi-threading, and amount of available memory.

Research has shown that students have difficulties identifying the base case of the recursive function (Haberman et al. 2002). However, very little is known about whether difficulties with base cases influence whether students choose to use recursion to solve algorithmic programming tasks. Investigating this issue is one of the purposes of this study.

2.3 Why knowing about students’ thinking about recursion matters

Research about the learning of recursion is of critical importance as recursion is a useful problem-solving tool used in programming. In addition, recursion is typically taught during introductory programming courses, which are usually the first courses in the Computer Science degree curriculum. The ACM curriculum suggests four academic classroom hours dedicated to the topic of recursion during the first programming course for beginning programmers. Research shows that 30 to 40% of students who enroll as a Computer Science major leave the degree program after taking their first Computer Science course (Cohoon et al., 2008). For a substantial percent of students, it appears that experiences in their first computer science course either leave them unprepared for subsequent courses or discourage them from pursuing the degree for other reasons.
This is problematic because the demand for computing jobs is rising at the same time that the number of students enrolling and graduating with a Computer Science degree is declining (Beaubouef & Mason, 2005; Cohoon, Wu, & Luo, 2008; Powell, 2008; Sloan & Troy, 2008). In addition to a decline in enrollment, the gender balance in Computer Science is skewed with more males pursuing the degree (Cohoon et al., 2008). If we would like to match the demands of the computing force job, we need to try to change the attrition rate of students from computer science. A focus of these efforts should be on the introductory programming course given the significant role it appears to play in students’ decisions about pursuing the degree. One approach to improving success and retention rates is to improve student understanding of important ideas. Therefore, it may be useful to investigate one of the most difficult topics taught within the first programming course—recursion.

The primary goal of this study is to identify when students use recursion as an approach to solve various programming tasks. As mentioned previously, the ACM curriculum states that recursion is a topic that should be introduced during the first programming course. However, it reappears during the upper level courses, in the form of recursive algorithm and recursive structures. Therefore, recursion is an important topic throughout the curriculum. If we would like for students to succeed and complete degrees in Computer Science, it may vital for students to understand the concept of recursion and to know how to construct recursive solutions. Thus, knowing what makes programming topics, such as recursion difficult for students, is useful subject to pursue.
2.4 Recursion is important and difficult

As noted above, recursion is considered to be one programming concept that is difficult for students to learn. Findings from various research studies suggest that recursion is also a hard topic to teach as an educator (Bruce, Danylik & Murtagh 2005; Dann, Cooper & Pausch 2001; Ford 1982; Turbak, Royden, Stephan & Herbst 1999; Velazquez-Itrubide 2000). Computer Science educators have described the process of teaching recursion as a challenge, largely because students have difficulties envisioning the abstract concept (Dann, Cooper & Pausch, 2000). Previous studies revealed difficulties of first year students with understanding the concept and applying it in order to solve programming problems. It has been documented in many studies that students have difficulties using recursion as a tool to solve problems. Haberman & Averbuch (2002) found that students have difficulties in identifying base cases. Students might include redundant base cases, ignore boundary values and create degenerated cases. Students might avoid out-of-range values, and they may even not define any base cases when formulating recursive algorithms.

Some tasks may be easily solved using recursion while others can be solved using iteration. Recursion is considered to be a fundamental concept in programming (Kurland & Pea, 1985; Turbak & Royden, 199AD; Velázquez-Itrubide, 2000). It can be fast, elegant, and convenient when solving programming problems. Every recursive solution could be implemented using iteration and vice versa; however some iterative solutions can be complicated whereas the recursive solution is simpler and elegant.

Because iteration is considered to be a natural approach as one is repeating the same idea, solving problems recursively requires students to expand the way that they
look at problems and consider them from another perspective. Recursion is considered a top-down approach to problem solving, where solving smaller and smaller instances of the original problem will solve the overarching problem. Research has shown that students often find using the new technique, the recursive solution, difficult to implement.

Thompson (1985) offered a hypothesis for why students are having extreme difficulty with recursion. Thompson (1985) claims that in order to be able to write recursive procedures or functions, one must first learn how to describe the object as a recursive structure. This means a student must identify the sub-problems prior to solving the complete problem. For example, when counting the nodes in a binary tree, one must identify the binary tree as a recursive structure, in order to write a recursive procedure.

A secondary goal of this study is to investigate how students define the task. It could be that only students who can define the task in a recursive way choose to solve the task recursively, as Thompson suggested. As an addition to Thompson’s research, this current study will examine how students define the task and whether students solve these tasks using recursion.

Ginat (2004) found that students do not turn to recursion as a problem solving approach when solving programming tasks. Even though when learning about recursion students often asked to employ recursion, upon approaching a programming task, a computer scientist is not told which approach to use. Ginat conducted a study with 23 college-level students, who completed various courses during which they were introduced to and practiced solving programming tasks using recursion. The students were given a variety of tasks to solve (using their preferred programming language).
Their solutions were examined. Some students were interviewed about their solutions. Giant’s research discusses and presents findings from three tasks and the analysis focused on examining problem solving approaches when no explicit hint of direction is provided. Ginat’s tasks differ from the interview tasks given to participants in the current study. The first task is Giant’s study involved counting of the number of different ways to reach a basketball score. The second task involved counting the numbers of distinct elements in a list of numbers. The third task asked to minimize the total number of mathematical in order to produce to two numbers, within giving limits. These tasks are considered by Ginat to be optimization and counting, which are common and fundamental in various CS domains. This current study did not include any optimization task or a counting task. The tasks used in the interview will be discussed in Chapter 3.

Ginat concluded that that for all the three tasks, the majority of the students did not invoke or follow the recursive approach. Instead, some participants chose to use iteration, which led to unsuccessful solutions. Little information is provided in Ginat’s study regarding data collection and analysis, therefore drawing specific parallels between the design of the current research and Ginat’s research is not possible.

The goal of the current study is to investigate if students choose to use recursion as their initial approach when solving programming tasks. As an addition to Ginat’s findings, the aim of this study is to investigate whether students can actually successfully solve the tasks using recursion even if that approach was not their first choice for a solution.

2.5 Teaching recursion is challenging
Educators should teach recursion in order to give students all the tools necessary to solve problems using this concept. However, some computer science educators have described the process of teaching recursion as a challenge largely because students have difficulties envisioning the passive flow of the recursive function, where the function builds up the solution by using solutions to other instantiations (Dann et al., 2000). Additionally it was found that students do not gain an understanding of the passive flow, therefore they have difficulties constructing and understanding the concept of recursion (Scholtz & Sanders, 2010). Wirth (2008) claimed that recursion is often introduced as the process of an algorithm when one of its steps involves the re-running of the same algorithm. Wirth found that this can often be confusing to students. For example, in order to compute 5! recursively, one will need to compute 4!, which can be confusing since one is re-running the same algorithm until reaching the base case.

Researchers have shown that teaching recursion while using visual and graphical examples can help students to understand recursion and motivate them to use it when solving programming problems (Stephenson, 2009). From his study, Stephenson created three sub-goals to help in teaching recursion. First, the problem should be interesting to the students. Secondly the problem should serve a practical purpose. Thirdly the problem should be as easy to solve using recursion as it is with iteration, in order to motive students to use recursion. Stephenson (2009) claims that by using examples that reach the three main goals, students are more engaged and motivated to learn and use recursion while solving problems. Stephenson does not provide any evidence to support his claim, which is an important limitation of his study. Although Stephenson reports positive outcomes from teaching recursion in these ways, the work did not provide the computer
science education community with insights into why this approach was effective and what specifically it helped students better understand about recursion.

Hsin (2008) used the example of Towers of Hanoi, which is a game widely used to illustrate the concept of recursion. When teaching recursion using the Tower of Hanoi example, a visual graph was presented. Hsin (2008) suggests that using a visual aid may help students understand the flow of a recursion process. In other words, by using visual aids students can understand the recursive call in addition to the passive flow, where a value is passed back to previous invocations and a solution is built. This effort, however, did not provide detailed insights into why these visual aids might help students develop a robust understanding of recursion or how this approach influenced student thinking about recursion.

To summarize, teaching recursion can be challenging as an educator (Bruce, Danyluk & Murtagh 2005; Dann, Cooper & Pausch 2001; Ford 1982; Turbak, Royden, Stephan & Herbst 1999; Velazquex-Itrubide 2000) however, research has shown that using examples can increase students’ understanding of the concept of recursion, in addition to employing recursion as a tool to solve a certain task. Although researchers and educators proposed various approaches to improve the teaching and learning of recursion, these efforts have not generated insights into why the approaches were effective in helping students develop strong understanding of recursion.

This current study helps provide insight into students’ thinking about recursion, which may help improve teaching practices that can motivate students to use recursion and construct better understanding of the topic.
2.6 What do novice and non-novice programmers know about recursion and what are their difficulties?

Recursion is a difficult concept to learn as a novice and it seems to be logical that experts will know more than novices. However, what do novice programmers really know about recursion? What do novice programmers understand about the behavior of the recursive procedures? The ideal knower should know the three features of recursion. They should know about the active flow of the recursive function, also called the recursive call, the base case and the passive flow, which is where a value is passed back to previous invocations and a solution is built.

Kahney (1983) showed that most novice programmers do not understand the element of the passive flow of recursion. However, some of the novice programmers acquired and understood all three elements of recursion. This means that most students do not understand all three features of recursion. They preserve the recursion calls as one instant that is repeating itself in order to solve the task. In other words, they do not understand that the recursive call is a sub-problem that gets executed as its own function.

When it comes to the first programming course, Lahtinen et al. (2005) claim that either novice programmers learn the syntax of the language, functions, procedure and recursion fairly easily, or have problems with it all. In other words, students either understand the programming language that they are programming in or they understand the concept of recursion, but rarely are they able to understand both the programming language and recursion well enough to solve programming tasks using recursion.

Literature shows that novice programmers have difficulties with the concept of recursion, but do advance programmers, such as computer science seniors, have problems
with recursion as well? Ginat (2004) investigated computer science seniors and their understanding of the concept of recursion. Ginat examined whether senior computer science programmers turn to recursion as a problem solving heuristic. His results show that students do not use the recursive approach when solving programming tasks. Instead, they attempted various iterative solutions and were sometimes unsuccessful in generating a correct solution. Ginat focused on problem solving using recursion and claimed that in order to successfully solve a problem using recursion, one should think of the problem with backwards reasoning. Backwards reasoning is when one is defining the base case first, and then thinking about what values need to be passed back to previous invocations in order to built a correct solution. Backward thinking was also defined by Götschi et al., (2003, 2006) as passive flow.

Researchers have demonstrated that recursion is challenging for novices as well as more advanced programmers in computer science. These findings, however, do not provide many insights into why this is the case or what it is about recursion that is persistently difficult in these ways.

2.7 When should recursion be taught?

The ACM 2008 curriculum suggests a minimum of four academic hours covering the topics of recursive mathematical functions, simple recursive functions and recursive structures. Recursion is usually taught in the introductory programming course in computer science. During the first computer science course, students are introduced to iterative approaches such as for loops and while loops, before the notion of recursion is introduced. These kinds of loops are frequently used in conjunction with iteratively-
structured solutions to programming problems. Students find for loops and while loops very intuitive (Stephenson, 2009). Using recursion requires students to expand their way of thinking and look at a problem from a different perspective. Students must identify the base case, where to implement the recursive call, and where and how a value will be passed back to previous invocations in order to construct a correct solution.

When reflecting upon their teaching, Levy and Lapidot (2000) suggest that introducing recursion after introducing iteration may impact students’ learning of recursion. Levenick (1990) argues that recursion should be taught prior to iteration due to the fact that the topics are interchangeable and he believes that iteration prevents conceptualization of recursion. Both these claims by Levy, Lapidot and Levenick are not grounded by strong evidence. In examining similar issues, Wiedenbeck (1988) found that there was not much difference in understanding when students were exposed to recursion before iteration and vice versa. She did find that repeated examples and practice were effective in developing students’ understanding of the topic.

From these findings, there is no clear conclusion about whether the order of the instruction influences students’ learning of recursion.

2.8 When Do Students Use Recursion And Why?

Even though researchers agree that recursion is important, should be taught, and can be a powerful problem-solving tool, we have yet to understand when and why students use recursion when solving programming tasks. Little is known about reasons why students do not use recursion as a problem-solving tool.
Ginat’s (2004) research showed that students use iteration when solving programming tasks, however Ginat’s study did not aim to answer why that is the case. Little is known about what solution will students choose if presented with an iterative solution and a recursive solution. Part of the current study was designed to investigate which solution students prefer when presented with iterative and recursive solution and their reasoning about this choice. Discovering why students chose to use recursion, as a problem-solving tool, is a useful in order to understand what type of difficulties students encounter. If we know students’ reasoning for their preferences when given a choice between a recursive solution and an iterative one, perhaps we can teach struggling students successful ways of thinking about these particular problems so they select recursive solutions when they are the better option for solving the problem.

Thompson (1985) claims that in order to be able to write recursive procedures or functions, one must first learn how to describe the object as a recursive structure. The current study includes an investigation of whether students describe and solve the task using a recursive approach and why do they choose to do so.

2.9 Summary

Research has shown that recursion is a hard topic to teach (Bruce, Danyluk & Murtagh 2005; Dann, Cooper & Pausch 2001; Ford 1982; Turbak, Royden, Stephan & Herbst 1999; Velazquex-Itrubide 2000) and hard to learn as a student (Anderson, Prirolli & Farrell, 1988; Dicheva &Close, 1996; Gotschi, Sanders & Galpin, 2003; Haberman &Averbuch, 2002; Kahney, 1989; Kessler &Anderson 1989). Recursion is an abstract concept that is vital to understand and to apply as a computer scientist.
Even though Giant (2004) has shown that students do not turn to recursion as a problem solving technique, the current investigation adds to the computer science education field by examining the question of why students choose to use recursion, or why they do not choose to use it.

We might be able to improve our teaching practices if we know when and why students choose to use recursion when solving programming tasks. If we can improve our teaching of the topic, it might be a less difficult topic for students. This might then improve how student perceive the first programming course and the computer science degree, which may lead to better retention rates within the degree program.
CHAPTER 3 - METHODOLOGY

3.1 Methods of data collection

3.1.1 Subjects of research. During the spring semester of 2012, 17 students from different disciplines at a large public university in the northeastern U.S. participated in one-hour long semi-structured clinical interviews (Hunting, 1997). Participants were a combination of three Computer Science sophomores, eight Computer Science juniors, two Computer Engineering seniors, three graduate students in Computer Science and one graduate student in Computer Engineering. Fourteen out of 17 participants received extra credit in their Computer Science classes in exchange for participation in this research. Names used in this paper are pseudonyms in order to keep the identity of the participants confidential.

In order to investigate the research questions, data included students’ verbalized thought processes and their problem solving approaches. To collect these specific data, an interview protocol was developed. The protocol was designed to generate data on what methods students use when solving programming problems, and to investigate the reason for choosing such an approach. The researcher developed both the interview protocol and the interview questions based on previous research. For example, the factorial task and the summation of digits task, which are discussed later in this chapter, were a part of research conducted by Haberman and Averbuch (2002) in a study regarding students’ difficulties with different parts of the recursive functions.

The participants in this study were enrolled in a higher-level Computer Science course. The course enrolled a combination of undergraduate and graduate students. The
course was taught simultaneously to both undergraduate and graduate students however, workload and assessment varied for the two populations. According to the curriculum of the institution where the study was conducted, in order to enroll in a higher-level Computer Science course, students should first pass a programming course. Hence, all participants of this study were taught and had practiced programming using recursion in a course setting prior to participating in this research.

3.1.2 Interviews. The interviews were semi-structured clinical interviews (Hunting, 1997). The tasks were prepared in advance and follow-up questions were asked in order to prompt participants to verbalize their reasoning when approaching tasks using their chosen method. All the interviews were audio-recorded and transcribed.

3.1.2.1 Background questions. Prior to solving the tasks, the participants were informed of the confidential nature of the interview. The participants were informed that they could include written work and could refuse to answer any questions during the interview if they wished to. In addition to this information, participants were asked to specify their level of education, major, and their programming background. Participants were also informed that they could solve the programming tasks using pseudocode or their preferred programming language. The interview consisted of five main parts as described below.

3.1.2.2 Part 1. The first part of the interview included three different programming tasks where participants were asked to write a function to solve a task. The primary goal of this part was to investigate the participants’ first solution to these tasks.
without prompting them to follow a specific approach. Each task included an example.

The three questions are presented in Figures 3.1, 3.2 and 3.3.

### Figure 3.1. Power Task

Write a function that for any given numbers x and n, the function will compute \( x^n \). The input would be two positive integers x and n, and the output will be the result of \( x^n \).

For example:

\[
\text{if } x = 4 \text{ and } n=3 \text{ the result will be } 4^3 = 64.
\]  

### Figure 3.2. Summation of Digit Task

Write a function that for any given number will return the summation of its digits. The number given is a non-negative integer.

For example:

\[
\text{If the number given is 9876, the function will return } 30. \quad (9 + 8 + 7 + 6 = 30)
\]

### Figure 3.3. The List Search Task

Given a number (N) and a list (L), write a function that will return True if the number (N) is in the list (L) otherwise return False.

For example:

- If the number given is 5, and the list is 1,2,5,4,3 the function will return True.
- If the number given is 7, and the list is 1,2,5,4,3 the function will return False.

After participants wrote down or discussed each solution to tasks 3.1, 3.2 and 3.3 separately, the researcher asked them to talk through each solution again. Some
participants restated the solution verbally while others used the given example or an arbitrary example, while talking through their solutions. Using an example can be helpful in order to track the state of the different variables and the output of the function. The goal for reiterating the solution was not only for the participant to repeat his or her answer, but also for the researcher to gain further insight into the participant’s thinking and why the participant chose a certain approach to solve the question. In addition, while participants reiterated the solution, the researcher gained a better understanding of their solution. After participants explained his or her answer, the researcher asked some follow-up questions. Follow-up questions were used in order to achieve further insight into participants’ thinking about approaching the tasks using recursion or iteration. In order to further investigate the participants’ responses, certain interview tasks were followed by particular follow up questions.

Task 3.1, Power Task, was a straightforward mathematical task that did not require any follow-up questions. However, if participants chose to solve the task using preexisting library functions (e.g., calling a function that gives the solution without having to create the code for that function), the researcher followed up by asking them to implement the function in order to solve the task. Because the goal of this study was to understand how students solve programming tasks and what approach they use to do so, if participants used a preexisting function they might not think about what approach is needed in order implement a function to solve the task. Therefore, the researcher asked the participants to implement a function to solve 3.1, without using preexisting library functions, in order to gain insight of what approach the participants chose in order to solve the task.
Task 3.2, The Summation of Digit Task, asked participants to write a function to calculate the sum of the digits within a positive given integer. This task was also a straightforward mathematical task; however, one might experience some difficulties when trying to break the number into individual digits. After task 3.2 (Summation of Digits), the researcher asked the participant if he or she considered modular arithmetic when solving the task initially. Using two different modular arithmetic functions (mod 10 and div 10), the participant could break down the number into digits and then add the digits, which would yield a correct solution.

Modular arithmetic is a system of arithmetic of integers. The function mod, standing for modulo, will compute the remainder of the division. For example, mod 10 will yield the units digit from any number and if that number is 5698, 5698 mod 10 will yield 8. The function div, standing for division, will compute the whole division without the remainder, even if there is a remainder, the function will yield an integer. For example, div 10 will result the original number without the unit digit and if that number is 5698, 5698 div 10 will yield 569. Since the length of the number is unknown in this task, one can truncate the number by taking away the units digit, where the tenths place is now in the units place, creating a new number. If this process continues, one can add up the digits from the least significant place to the most significant place until there are no longer digits to add. This is the simplest way to break down a number into its digits, if the length of the number is unknown.

Since the research questions focus on reasons for choosing a certain method while solving programming tasks, it was not the aim of the study for the participant to dwell upon breaking down the number to its digits. The primary goal of introducing whole
division and modulus is to assist participants with breaking the number into its digits and to investigate what approach would they use to add up the digits. After introducing div and mod to the participants, a follow up question asked the participants if they would like to change their initial solution or if they were content with it.

Task 3.3, also called The Search List Task, asked the participants to write a function that will yield “true” if the item is in the list and otherwise yield “false.” The example in the task was of an unordered list. After participants wrote down or discussed their initial solution, a follow-up question asked them to consider an ordered list, which might trigger participants to consider a different solution. A solution to finding an item in an ordered list may be implemented by using iteration rather than recursion. An item in an ordered list can be found using several different algorithms. One of the algorithms is called Binary Search. Binary Search compares the input item with the middle element of the list. If the values match, then the algorithm should return “true.” Otherwise, if the sought item is less than the middle item, then the algorithm repeats its action on the sub-list to the left of the middle item or, if the input key is greater, on the sub-list to the right. This algorithm can be implemented using recursion or iteration, however binary search is typically implemented using recursion due to efficiency reasons. If a participant chose to use a binary search to find an item in a sorted list, the researcher asked about the nature of the binary search implementation. That is, the researcher asked the participant how he or she would implement the binary search as a solution. The goal of this follow-up questions was to investigate if an ordered list and an unordered list would result in a different solution which might result in a different approach to solving the task.
3.1.2.3 Part 2. The second part of the interview consisted of two tasks, where participants were asked to examine two different solutions and “choose one and explain why you chose that solution”. In other words, the researcher asked the participants to choose between an iterative solution or a recursive solution and discuss why in their opinion, the solution they chose was the best fit to solve the task. In some interviews, participants asked the researcher:” what do you mean pick, is one better then the other?”, in those cases the researcher probed the participants and asked them to choose “which solution was a solution they would prefer when solving such a task or which solution would they produce if none of the solutions were shown“ to them. After the task was stated, an example was presented followed by a solution on the left and another solution on the right. One solution was an iterative solution and one solution was a recursive solution. It is important to note that both solutions solved the task successfully. Examples are shown in figures 3.4 and 3.5. A follow-up question asked the participants: “why did you choose this solution”. In particular, the researcher requested the participants to discuss why they chose the iterative solution or the recursive solution. The goal of this part of the interview was to evaluate and reveal reasons why participants choose a certain approach to the tasks when both types of solutions are given. The follow-up question prompted the participant to verbalize the reason why he or she choose to approach the task with an iterative approach or with recursive approach, and why they did not choose the other approach.
Write a function that for any given number n, the function will compute n! (factorial). The input would be a positive integer n, and the output will be the result of n!

For example:

If the number given is 5 (n=5), result will be 120

Consider the following solutions:

```
Factorial (n)
int Fact = 1
While (n ≠ 0) do
    Begin
        Fact = Fact * n
        n = (n - 1)
    End
Return Fact
```

Figure 3.4. Factorial task

Write a function that counts the number of nodes in a binary tree

For example: The tree below should return 15.

Consider the following solutions:

```
Count (Tree)
int count = 0
Stack S
node type - X
push (S , Tree)
While (S is NOT empty) do
    Begin
        X = top(S)
pop(S)
count = count + 1
        If (X.left is NOT empty) then
            push (S , X.left)
        If (X.right is NOT empty) then
            push (S , X.right)
    End
End
return count
```

Figure 3.5. Binary tree task
3.1.2.4 Part 3. During the third part of the interview, participants were asked to solve the first three programming tasks (part one of the interview) again; however, if they had solved a particular task using iteration earlier, during this part they were asked to solve the same tasks using recursion and vice versa. At the end of this part, the participants were asked to discuss why they initially approached the tasks the way they did. Even though participants saw and examine two examples of a recursive solution in part two (the previous part of the interview), it did not alter the results in this part. Participants were not allowed to refer back to the two examples they saw during the second part, in order to produce a recursive solution in part three. Also they were asked why they did not think to use the alternative approach. For example, if a participant solved the first three problems using iteration, he or she was asked to solve them again using recursion, followed by a discussion of why they initially approached the task iteratively and not recursively. Examples are shown in figures 3.6, 3.7 and 3.8.

Write a recursive function that for any given numbers x and n, the function will compute $x^n$. The input would be two positive integers x and n, and the output will be the result of $x^n$.

For example:

if $x = 4$ and $n=3$ the result will be $4^3 = 64$.

Figure 3.6. Recursive factorial task

Write a recursive function that for any given number will return the summation of its digits.

The number given is a non-negative integer.

For example:

If the number given is 9876, the function will return 30. ( $9 + 8 + 7 + 6 = 30$)

Figure 3.7. Recursive summation of digits task
The goal of this part of the interview was to determine whether participants were capable of solving the tasks using both approaches. In particular, the goal was to determine if the participants could solve the tasks using recursion. It is important to know that participants could solve the tasks using recursion because if the participant could not solve the task using this method, then it might be concluded that the participant was not comfortable solving task using recursion and that could be the reason they did not use it in their initial approach to solving the task. However, if a participant could solve the task using both approaches, the follow-up questions could help reveal why the participant preferred one method to the other given that they were capable of solving the task either way.

**3.1.2.5 Part 4.** During the fourth part of the interview, participants were asked to discuss how they modeled each of the tasks. For the purposes of this study, “modeled” means that the participants were asked to discuss how they thought of the task prior to solving it, or how they defined the task for themselves. Each task in the interview could be modeled using an iterative model or a recursive model. For example, $x^n$ can be modeled iteratively as $x \cdot x \cdot x \cdot \ldots \cdot x$, n number of times, or it can be modeled recursively as $x \cdot x^{(n-1)}$. Another example is factorial, which can be defined iteratively, and
recursively. Five factorial can be computed iteratively as \(5 \cdot 4 \cdot 3 \cdot 2 \cdot 1\), or it can be defined recursively as \(5 \cdot 4!\).

The participants were shown two different note cards with the two different models of the question. The goal for this part is to determine if there is any relationship between the way the participants modeled or defined the task and their initial solution to the task.

The data fell into two categories, iterative approach or recursive approach for each question. A comparison was made between these responses and part one and two of the interview.

**3.1.2.6 Part 5.** Lastly, participants were asked to define iteration and recursion and to describe the differences between them. In addition to defining the concepts, the participants were asked to explain iteration and recursion to a novice programmer using examples. The goal for this was to gain further insight to the participants’ thinking about recursion and iteration and the difference between them, which can help answer the research questions.

**3.2 Methods of data analysis**

Interview data were analyzed using an approach created by the researcher, inspired by Grounded Theory (Glaser & Strauss, 1967). The process of analyzing the data included searching for patterns related to the researchable questions within the audio-recorded interviews, the transcriptions of the interviews and the written responses to the interview tasks.
Grounded Theory is a method for developing a theory that is grounded in the data. In classic grounded theory, one should not review the literature before analyzing data. As the researcher reviewed the literature in order to create the interview tasks, which occurred prior to the data analysis phase, one can claim that this study was inspired by grounded theory but did not strictly follow the grounded theory methodology. In addition to reviewing the literature to create the interview tasks, hypotheses and categories were formed prior to the data collection phase in order to formulate interview protocol and follow up questions.

Each part of the interview resulted in different responses, which resulted in different analyses, all of which are described in this section.

3.2.1 Horizontal analysis. The first analysis is referred to as a horizontal analysis. Horizontal analysis can also be described as analyzing the data task by task. In particular, the researcher first analyzed written work of each task separately for all participants in the study. If a participant did not include any written work for a certain task, the researcher listened to the audio of the interview, while reading the transcripts in order to understand participants’ response. The first three tasks of the interview (computing the power of two integers, computing the sum of digits and searching an item within a list) were analyzed using the same algorithm. The researcher designed the analysis algorithm based on the various solutions of the interview question, the goal of the study and creating additional categories base upon the data. The algorithm development is described in this section. Responses from part five of the interview were analyzed. However, part five did not generate useful data that could help answer the research questions. Due to a lack of clear patterns within the data, and additional insight,
responses from part five will not be presented in this thesis.

3.2.1.1 Development of analysis for part one of the interview. Two major categories were formed prior to the data collection phase. Those two categories were influenced by the design of the study. The researcher knew that each of the first three tasks could be solved using either an iterative approach or a recursive approach. The goal of creating these two major categories was to answer the research questions about which approach participants preferred when solving the interview task. In addition to approach categories, the researcher defined correct and incorrect categories as well prior to data collection. It was fundamental to the study to investigate whether participants could successfully solve these tasks using both approaches. Therefore the two major categories, iterative and recursive approaches, in addition to the correct and incorrect subcategories, emerged from the design of the study by the researcher.

During the data analysis phase, two subcategories emerged. Within the iteration approach category, participants used two different iteration loops structures for successful and unsuccessful responses to the interview tasks. The researcher did not anticipate this; therefore two subcategories under iteration emerged during the analysis. This part was influenced by grounded theory; the subcategories emerged from patterns within the data. The next section describes the categories and their application to the data in detail. Figure 3.9 is a visual representation of the categories of the data, followed by a detailed description of the algorithm.
To conduct the analysis, the first step was to group similar data for each of the interview tasks. During this step, the researcher divided the written work (or the verbal solutions) into two different groups. The first group was categorized as solving the task using an iterative solution and the second group was categorized as solving the task using a recursive solution. The goal of dividing the data into iteration and recursion groups was to quantify how many participants chose to solve the task using a certain approach without prompting. This information helped answer the researchable question of what approach participants prefer to use when solving programming task.

An iterative solution can be defined as solving the problem in a repetitive way. In particular, one is repeating the same process with the aim of solving the problem. The use of repetition within a computer program is usually in the form of a loop. For loops
and while loops are two approaches classified as iterative approaches. A recursive solution is defined as solving the problem by dividing it into smaller pieces of the same problem. In a computer program, a recursive solution can be defined as a function that calls itself with a smaller domain, in order to achieve the solution to the whole task. Since iterative solution can be of the form of a while loop or a for loop, the researcher divided the group of iterative solutions into two different subgroups. One subgroup was for loop solutions and the other subgroup was while loop solutions. After groups and subgroups were formed, the researcher checked each solution for its correctness (based on criteria described below). During this step, subgroups of correct and incorrect were formed within the major groups described above. The figure below represents a correct iterative solution using a for loop to compute the power of two positive integers.

```
Power (int n, int x)  
    int Result = 1;  
    int i = 0;  
    For i to n do  
        Result = Result \cdot x;  
    Return Result;
```

Figure 3.10. A correct for loop solution to the Power Task

The first line in the solution (Fig. 3.10) represents the name of the function (power) with two input variables (x and n, both integers). The next step is to initialize two local variables Result and i. In this particular case, one of the local variables, Result, is initialized to 1 and the other variable, i, is initialized to zero. The input
variables $x$ and $n$ are local variables created on the stack. Once the function Power is called with two numbers, these values are set to $x$ and $n$ respectively. For example, Power (4, 2) will set $n$ as 4, and $x$ as 2. After creating and initializing the local variables, the next step is to implement the repetitive multiplication using a \texttt{for} loop, which is the iterative part. The \texttt{for} loop structure will increment $i$ by one for each iteration through the loop. In general, \texttt{for} loop structures can be described as a conditional statement of the numeric range from lower boundary $i$ to the upper boundary $n$. Within the \texttt{for} loop, the indented command, $\textbf{Result} \textbf{equal} \textbf{Result} \textbf{multiplied by} x$, will calculate the value of $\textbf{Result}$ multiplied by $x$, and will store it in the variable $\textbf{Result}$. Each time through the loop, the variable $\textbf{Result}$ will be updated by the new value. As $i$ is set to 1 at the beginning of the loop, the statement after the \texttt{for} loop will be repeated until $i$ reaches $n$. As $i$ is incremented by 1, it will reach the value of $n$ and the program will return $\textbf{Result}$, which is the last statement of code in the figure. The flow chart below describes the iterative solution of the Power Task.

Using a \texttt{while} loop can yield a correct solution as well. The difference between \texttt{while} loop solutions and \texttt{for} loop solutions is the stopping condition of the loop. In any loop, a local variable will keep incrementing or decrementing until a condition is reached to break from the repetition. In a \texttt{for} loop, when the lower boundary reaches the value of the upper boundary, statements within the \texttt{for} loop will not be executed and the program will continue with the next line of code after the loop. In a \texttt{while} loop, statements will execute until the condition within the \texttt{while} loop is no longer true. For example, when searching for an item in a list, one can keep looking at the first item in the list and removing it until the list is empty or the item is found. Therefore, the \texttt{while}
loop condition will be, while the list is not empty and the item is not found, the search will continue. In a \texttt{while} loop, the programmer needs to change local variables, etc., so that the termination condition will ultimately be met. However \texttt{for} loop does that automatically for the programmer. In some languages, \texttt{for} loop statements allow the programmer to decide if the local variable, \( i \), will be incremented or decremented and what that value will be. Usually when writing \texttt{for} loop, the local variable \( i \) will be incremented to decrement by one.

Figure 3.11. Flow chart of an iterative for loop solution to the Power Task
3.2.1.2 Application of analysis to part one of the interview. First, the researcher focused on all the responses from each task. For each task, the researcher divided the data into two groups, iteration and recursion. Two examples of iterative responses are presented in Figures 3.11 and 3.12. A recursively structured solution is presented in Figure 3.9.

If a response included `for` loop or a `while` loop structure, the researcher decided it would be grouped within the iteration category. If a response was in a form for a recursive function, the researcher decided it would be grouped within the recursion category.

Second, the researcher divided all the responses in the iteration category into `for` loop and `while` loop responses. If a response was structured as a `for` loop, such as the example in Figure 3.10, it was categorized as being in the iteration category within the category, `for` loop. If a response was structured as `while` loop, such as the example in Figure 3.12, it was categorized as being in the iteration category within the category, `while` loop.

```
Power (int n, int x)
    int Result = 1;
    While (n != 0)
        Result = Result * x;
        n = (n-1);
    Return Result;
```

Figure 3.12 A correct `while` loop example to the Power Task

Third, the researcher needed to distinguish between correct responses and incorrect responses. In order to do so, the researcher used the flow chart from figure 3.3
to investigate the correctness of the for loop responses. A similar flow chart was constructed in order to investigate the while loop responses.

The researcher investigated correct and incorrect recursive responses using a table to indicate the state of each variable. An example of such a table is presented as table 3.1. This one is for implementing the iterative solution to the Power Task, using an example of $4^2$. The left column of the table shows the statement of the function, in their order of execution. All the columns to the right are presenting the variables and their current state with each line of the function.
3.2.1.3 Development of analysis of part two of the interview. After the responses to the first three interview tasks were analyzed (part one), the researcher analyzed part two of the interview. This included the forth and the fifth tasks, calculating factorial and counting the number of nodes in a binary tree, respectively. Because a correct iterative solution and a correct recursive solution were presented to the participants, there was no need to group the data into correct and incorrect categories. The data were divided into

<table>
<thead>
<tr>
<th>Command</th>
<th>Variables</th>
<th>$n$</th>
<th>$x$</th>
<th>$i$</th>
<th>Result</th>
<th>is $i = n$?</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (2,4)</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int $Result = 1$</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int $i = 0$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For $i$ to $n$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0 $\neq$ 2</td>
<td>(False)</td>
<td></td>
</tr>
<tr>
<td>$Result = Result \cdot x$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>$4 \cdot 1 = 4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For $i$ to $n$</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1 $\neq$ 2</td>
<td>(False)</td>
<td></td>
</tr>
<tr>
<td>$Result = Result \cdot x$</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>$4 \cdot 4 = 16$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For $i$ to $n$</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2 = 2 (True)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return $Result$</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
<td>Result = 16</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 Example of tracing table for calculating $4^2$
two different groups, based on participants’ choice of an iterative solution or of a recursive solution.

3.2.1.4 Application of analysis to part two of the interview. The researcher counted the number of participants who chose the recursive solution verses the number of participants who chose the iterative solution. During the interviews, participants were asked to circle the solution of their choice; therefore a circled solution was considered the written work of this part of the interview. If a solution was not circled, the researcher listened to the audio of the interview to decide the choice of solution. If during this part of the interview, a participant said “iterative” or “iteration” the researcher decided it would be counted as an iterative choice category. If a participant said “recursive” or “recursion” the researcher decided it would be in the reclusive choice category. The goal of this was to quantify how many participants chose recursive solution to the programming task when presented with both an iterative solution and recursive solution. This information can be helpful to investigate whether participants responded differently when asked to solve the task without prompting and when presented with two different solutions. After the participants chose a certain solution they were asked why they chose that solution, and from those responses, categories were developed. After analysis was complete, five dominant response categories became apparent: “stack overflow,” “properties of recursion,” “structure resemblance,” “lack of confidence in solution,” and “other”. These categories and criteria for placing students’ responses in them are described in Chapter 6.
3.2.1.5 Development of analysis for part three of the interview. Part three of the interview was analyzed horizontally as well. During this part of the interview, participants were asked to solve the first three tasks again but this time they were prompted to use the other approach. For example, if a participant initially solved the first task using iteration, this time he or she was asked to solve it using recursion and vice versa. These data were analyzed using the same algorithm as was used to analyzed data from the first part of the interview. The researcher grouped the responses into two major groups: iteration and recursion. Within iteration there were subgroups of for loop and while loop solutions. Correct and incorrect groups arose as well, for each group and subgroup. These groups are identical to the groups and subgroups described for part one. A correct iterative solution using a for loop was presented in Figure 3.10 and a correct recursive solution is presented in Figure 3.13.

```
Power (int n, int x)
    If (n=0) then
        Return 1;
    Else
        Return (x \cdot Power(n-1, x));
```

Figure 3.13 A correct recursive solution to the Power Task

The difference between the iterative example and this example is that repetition does not occur inside the function; instead the function is called within itself, until it reaches the breaking condition, which stops the recursion and returns a single value to the previous call. When the function is called with certain inputs, the first commend is an if
statement called the base case. A base case is a conditional that will break from the recursion. In this example, the base case is when the value of n is zero. Once n is zero, the recursion will stop and return 1 to the previous call. If n is not equal to zero, x will be multiplied by calling the function Power again with x and n -1. The flow chart in Figure 3.14 describes the recursive solution of Power Task using the example of x = 4 and n = 2.

After solving all three tasks using the other approach, participants were asked to verbalize why they initially chose to solve the task using the first approach. From those verbal responses, categories to answer the researchable questions were developed. Four dominant response categories became apparent: “stack overflow,” “class or teaching related reasons,” “lack of confidence in solution,” and “other”. These categories and criteria for placing students’ responses in them are described in Chapter five.

3.2.1.6 Application of analysis to part three of the interview. Part three was analyzed similarly to part one. During this part, participants were asked to solve the first three tasks with the approach they had not used previously. The division of the data into categories and sub-categories was discussed in Chapter 5. Responses to part three of the interview included various responses related to students’ creation of recursive solutions to the task. Since there were no sub-categories under the recursion responses, only correct and incorrect categories were necessary. In order to investigate the correctness of recursive responses, the researcher used a flow chart to follow the recursive function in addition the states of each variable.
Figure 3.14 is a validation of the recursive solution using the example Power(2, 4). Once the base case is reached, a value will be returned to the previous call, until the first call has been reached, which will output the final value. The curved arrows in the example below represent this process.

Figure 3.14 Flow-chart for recursive example of the Power Task.
3.2.1.7 Development of analysis for part four of the interview. During the fourth part of the interview, participants were asked how they initially thought of the task prior to solving it. In particular, the researcher was interested in how the participants thought of the key features of the task. Each participant was presented with two different options, one was an iterative representation and one was recursive representation. Figures 3.15 and 3.16 show the iterative and the recursive options to the Power Task respectively. Figure 3.17 and 3.18 represent the iterative and the recursive options to the factorial task respectively.

![Figure 3.15 An iterative definition of the Power Task](image)

![Figure 3.16 A Recursive definition of the Power Task](image)

![Figure 3.17 An iterative definition of the factorial task](image)

![Figure 3.18 A recursive definition of the factorial task](image)
The data for this portion of the analysis were participants’ choices between the iterative approach and recursive approach for each question. The consistencies between part four responses and part one and three responses were investigated. The goal of this research was to investigate when students use recursion to solve programming tasks and why. It can be speculated that if a student envisioned or defined the problem in a certain way, that could affect what approach he or she used when solving the tasks. For example, if a participant defined the task recursively and solved it using a recursive approach, it could be claimed that their definition of the task was the reason why they chose to solve the task recursively. The relationships between the definition of tasks and the responses to tasks can shed some light on the reason why students chose to use or chose not to use recursion as an approach to solve programming tasks.

3.2.2 Vertical analysis. The second analysis is referred to as a vertical analysis. Vertical analysis can also be described as analyzing the data participant by participant. This analysis was the same analysis as the horizontal analysis, however, instead of analyzing one task at a time, the researcher analyze the data one participant at a time. In other words, the researcher investigated collectively all the tasks for each participant. During this analysis for each participant, the researcher listened to the interview and read the transcripts, in addition to analyzing the written work. The practice of listening to the audio of the interview while reading the transcripts was something the researcher found useful but there was no particular analysis-based reason for this aspect of the process. While listening to the interview and reading the transcript, the researcher noted keywords participant used while solving each task. The goal of identifying keywords within the
interview transcripts and audio files was to determine whether a participant who did not include a written response verbalized an iterative solution or recursive solution. Only iterative keywords were found within the data. Keywords were, for example, “[to] iterate through” or “looping through.” These key words served as indications that the participants were thinking iteratively. Upon recognizing keywords, the researcher noted how the participant solved the given task. In particular, the researcher noted if the participant chose to solve the task using an iterative approach or recursive approach. If the participant chose to solve a task using iterative approach, the researcher categorized the type of loop used in the solution. After categorizing the type of approach, the researcher checked the solution for its correctness. This part was carried out in a manner identical to the corresponding part of the horizontal analysis. During this phase, the researcher did not reference the previous analysis. The goal of not referring to the previous analysis was to check the previous analysis to see if the data were categorized consistently and yielded the same results. After categorizing whether a solution was correct or incorrect, the researcher compared the horizontal analysis category of the response with the category assigned during the vertical analysis. The horizontal analysis and the vertical analysis were consistent and all the data were categorized correspondingly.

After categorizing the first three tasks, the researcher continued to the fourth and the fifth tasks, categorizing responses as iterative solutions and recursive solutions for calculating factorial and counting the nodes in a binary tree. The researcher categorized what solution the participant chose, in addition to the reason why this approach was chosen.
Results of both the horizontal analysis and vertical analysis were combined and are presented in Chapters 4, 5 and 6.
CHAPTER 4 – STUDENTS SOLVING PROGRAMMING TASKS

Written and verbal responses to tasks used during the five-part interview were the sources of data on participants’ approaches to solving programming tasks. One of the research questions is: Do students choose to use recursion when solving programming tasks? This chapter presents the findings from the first part of the interview (that was the source of data relevant to answering this question) and a discussion of answers to the research questions. The first part of the interview aim to investigate what approach do students choose to use when solving programming tasks, and the reasons for making that choice.

4.1 Students do not choose to solve programming tasks using recursion: results from part one of interview

During part one of the interview, participants solved three tasks without being prompted about which approach to use. As described in Chapter 3, data from these three tasks were analyzed using an algorithm that was developed by the researcher. The sections below present the findings from the first part of the interview, which includes separate discussions of the findings from data generated by each task. These findings show that when given programming tasks, without prompting for the use of a particular approach, participants do not use recursion. In particular, participants solved these tasks using iteration as a preferred method, even though these tasks could have been solved using recursion and, in fact, even when a recursive approach may be considered more intuitive. The first task is referred to as the Power Task and it asked participants to write a
function to calculate \( x^n \). The second task, called the Summation of Digits Task, asked the participants to sum up the digits of a certain given a positive integer. The third task asked participants to return true if an item is within a list of entities and otherwise to return false. These three tasks are discussed in detail within this chapter.

### 4.1.1 Power Task findings

For the first task, the participants were asked to write a function to calculate the result of raising one positive integer to the power of a second positive integer where both integers were given. This task is referred to as Power Task.

Initially, 13 participants out of 17 responded that in order to solve the Power Task, they would chose to use a library function to calculate \( x^n \). A library function is a built-in function, using pre-written code, that is included in the programming language. The library contains codes that can provide various functions to the users. For example, the calculation of an exponent is available to the user to use in order to solve the Power Task. After participants verbalized that they would use a built-in function, the researcher asked them to implement the exponent function without using pre-written code. The goal for this was to investigate what approach the participants would chose to use in order to solve the Power Task. When using the built-in library function, a participant may or may not have knowledge about how the built-in function is implemented. Therefore the researcher asked participants, who chose to initially solve the task using built-in function, to write their own solution without using pre-written function. Analysis was done on the participants’ written response that did not use a built-in library function.

To conduct the analysis, responses were first divided into two groups: solutions to the Power Task using an iterative approach and ones that used a recursive approach.
Horizontal analysis (i.e., analysis of all participants’ solutions to the task) revealed that 13 participants, who represent 76.5% of participants, solved the task correctly and 17.6% of the participants, represented by three participants, solved the problem incorrectly. Only one participant out of seventeen, who represents 5.88% of the participants, used recursion when solving this task of the interview and sixteen participants did not solve the first task using recursion. Three of the iterative solutions were incorrect. Figure 4.1 represents the findings from data on the Power Task.

**Figure 4.1 Power Task Results**

4.1.1.1 Iteration: For loop and while loop responses. For loops and while loops are two types of iterative loops. For loop is a statement that allows code to be repeatedly executed. A while loop is a conditional statement that allows code to be
executed repeatedly based on a given condition that can be true to false. When the condition is true, the 0 loop will keep executing the code until the condition becomes false. During the second part of the analysis (after solutions were coded as iterative or recursive), participants’ iterative responses were divided into for loop and while loop subcategories. Finding shows that when writing a function to calculate power, all participants who approached the task using an iterative approach used a for loop structure. In other words, results show that participants who solved the task using an iterative approach also used a for loop in their solution.

4.1.1.2 Correct responses The first step of analysis showed that one participant solved this task using a recursive approach. Further analysis led this response to be categorized as a correct response. To determine it was correct, the researcher tracked the response using the tracking algorithm described in Chapter 3. The researcher found that the recursive response did produce an output of $x^n$, therefore this was categorized as correct. Figure 4.2 contains written work of the correct recursive response to the Power Task.

```
for n = 0;
    return exp(x, n-1) * x
```

Figure 4.2. Correct recursive solution to Power Task
Initial data analysis showed that the five participants who chose to solve the task using an iterative `for` loop made programming errors. These participants made an error in the range of the for loop. In particular, these participants produced an output of $x^{n+1}$ instead of producing an output of $x^n$. At first the researcher marked these responses as incorrect; however, because this is a minor error that would not change the participants’ preferred approach, and could be corrected using additional tracking and additional examples, the researcher decided to group these responses in the category “correct with a minor error.” In addition, this error could have been avoided if they had done the task using a recursive approach. Figure 4.3 contains a participant response to the Power Task that was considered correct, although the response includes a minor range error as described in this section.

![Figure 4.3. Correct response (with minor error) to Power Task](image)

```plaintext
function (x, n):
    int result = x
    for n times
        result = result * x
    return result
```
It is important to state that one of the participants in this study choose to solve this task using the recursive approach and the response was determined to be correct. To summarize, none of the participants who chose to solve the Power Task using a recursive approach solved it incorrectly.

4.1.1.3 Iteration: incorrect responses. Responses were coded as incorrect when the output of the function did not produce an output of \( x^n \). This was determined by using a table tracing the state of each variable and output, as described in Chapter 3.

Two participants solved the Power Task incorrectly by overriding the variable \( x \) as it was multiplied by itself in their responses. Without using a local variable to hold the result of the calculation, the function overwrites the given variable \( x \) with a new value. For example, if the given task was to calculate \( 4^2 \), during the first iteration the function will calculate \( 4 \cdot 4 \), and restore this result as the new \( x \); therefore, the new \( x \) would be 16. The second time through, the function will calculate \( 16 \cdot 16 \), and store the result, which would be 256 and would become the new \( x \) and so on. This type of response does not produce an output of \( x^n \), therefore it was categorized as incorrect. Figure 4.4 shows a participant’s written work that would be categorized as an incorrect response as discussed above. The fourth command line shows overwrite of the variable \( x \).
The second type of incorrect solution that emerged from the data included a function that added the multiplication each time through the loop. This did not produce an output of $x^n$ and therefore was considered as incorrect response. For example, in the first iteration through the loop, $4 \cdot 4$ is added to the variable result, which was initialized to 1, therefore result is now holding the value 17. The second time through the iteration the function will multiply result by 4, which would be $17 \cdot 4$, which would be 68, and then would be added to result, which would be $17 + 68 = 85$. The addition to result is unnecessary and this yields an incorrect response that does not produce $x^n$. Figure 4.4 shows a participant response that would yield an incorrect response as discuss above.

4.1.2 Summation of digits findings. For the second task, the participants were asked to write a function to calculate the summation of the individual digits of an integer. Hereafter this task will be referred to as “the summation of digits task.” First, responses were divided into two groups: solutions to the task that used an iterative approach and ones that used a recursive approach. Horizontal analysis revealed that all seventeen participants, who represent 100% of the participants, used iteration when solving this task.
and none of the participants solved the summation of digit task using recursion. Figure 4.5 represents the findings from the summation of digits task.

Figure 4.5. Summation of digits task results

4.1.2.1 Iteration: For loop, while loop, and responses. The second part of the analysis was to divide the iterative responses into for loop and while loop subcategories. From the data, however, a new subcategory “other” emerged. Two participants’ responses did not fit into the for loop subcategory or into the while loop subcategory. Both participants solved this task using pseudocode notation and chose to write, “phase through the number” or “iterate through the number” as their solution. This
was considered to be an iterative solution, however one could not tell if this participant intended for it to be a `for` loop solution or a `while` loop solution. Therefore, a subcategory “other” under iteration was created.

Findings show that 11 participants chose to solve the task using a `for` loop, four participant chose to use a `while` loop and two participants used pseudocode which was considered as “other.”

4.1.2.2 Iteration: correct and incorrect responses. Of the participants who solved the task using some form of iteration, 15, who represent 88.3% of participants, solved the task correctly and 11.7% of the participants, represented by two participants, solved the task incorrectly.

Four participants, who solved the task correctly, chose to solve the task using an iterative `while` loop structure, incorporating modulus and whole integer division. Modulus and whole integer division are described in Chapter 3. Figure 4.6 contains written work of a correct iterative response using a `while` loop.

```
int summation (int num)
    int total = 0
    while (num != 0)
        x = num % 10
        total += x
        num = num / 10
    return total
```

Figure 4.6. Correct iterative solution to Summation of digits task using While loop structure
Out of 15 participants who solved the summation of digits task correctly, two participants used pseudocode notation in order to solve the task. Figure 4.7 contains written work of a correct iterative response using pseudocode notation. In addition to the written solution, the participant mentioned that he or she would “parse characters to digits using some kind of a loop,” therefore this solution was considered to be an iterative solution.

![Figure 4.7. Correct iterative solution to Summation of digits task using pseudocode notation](image)

Out of 15 participants who solved the summation of digits task correctly, nine solved it by using a for loop structure. In addition to using a for loop, these participants converted the number (an integer type) to a string or an array. When dealing with an integer, one must understand how to break it down into its digits using a recursive approach or an iterative loop. If one chooses to convert the integer to a string or an array, one can pull the individual digits from the integer (now stored as a string or an array) by using index notation. For example, if the number 9876 is now an array A, the first position in A or A[0] will contain the digit 9. The second position of A, or A[1] will
hold the individual digit 8. Therefore, while using index notation of an array or string, one can break the integer into its digits. Even though it might be trivial to break the integer into individual digits when using array or string index notation, one must know the length of the number in order to decide when to stop the iterative loop. One can compute the length of the number once, and store the result in a local variable; however, participants in this study calculated the length of the number each time through the loop while they re-computed the length. This is a high cost function, which can make the function very inefficient. In addition to finding the length of the string containing the number, when breaking the number into its digits using string index notation, the digit is not an integer type. The item pulled from the string is a character from the string, which may or may not be an integer or even the needed integer. If one is using string index notation to break the number into its digits, one must convert the character of the string into an integer in order to use the summation operation on that item. Characters of the string cannot be added to one another. Converting a character of a string to an integer is another high cost operation depending on the programming language used. This could be avoided by using recursion or modulus arithmetic and whole integer division. Figure 4.8 contains written work of a correct iterative response using a for loop structure. The second statement in the figure represents the conversion of the integer to a string. The second to last statement represents the conversion back to an integer. In addition to the conversion to a string and back to an integer, the for loop must calculate the length of the list each time through the iteration. The fourth statement represents this process.
Figure 4.8. Correct iterative solution to the Summation of digit task using for loop

```python
def summation(number):
    string = str(number)
    temp = 0
    for index in string:
        temp += int(index)
    return temp
```

Above, findings were presented from 14 out of 17 participants. The remaining three participants solved the task using an iterative approach in the form of a `for` loop structure. The responses also included breaking the number down using remainder division (modulus) and whole integer division, which will be referred to as the “division operations.” Two out of the three participants solved the task using a `for` loop and division operations and solved the task incorrectly, whereas the other participant solved it correctly.

Out of the participants who used an iterative `for` loop, one participant used the division operations the break the number into its digits, in addition to an iterative `for` loop. The written solution used the new truncated number without the most significant digit as the upper boundary of the `for` loop. In particular, the `for` loop will stop when the number is truncated to only the digit zero. For example, if the number was 543, the first time through the loop 5 will be added to the summation variable, than 43 would be stored as the new number. The second time through the loop, 4 will be added to the summation variable and 3 would be the new number. The third time through the loop, 3 will be added to the summation variable and 0 will be the new number, which will end
the loop. Figure 4.9 present a correct written response to the summation of digits task using `for` loop and division operations.

![Figure 4.9. Correct iterative solution to the Summation of digit task using `for` loop with division operations](image)

Two of the participants solved this task incorrectly using an iterative `for` loop structure in addition to using division operations to break the number into its digits. The reason to categorize these responses as incorrect is because the solutions did not output the summation of the individual digits of the given integer. One participant wrote an infinite `for` loop response. In particular, the upper bound of the `for` loop was omitted, which would have generated a syntax error in most languages. However, logically omitting the upper bound corresponds to an infinite loop. Figure 4.10 contains incorrect written work for the summation of digits task using a `for` loop and division operation. The third statement represents the `for` loop structure. The participant wrote $X$ as the upper boundary and that will never halt the `for` loop. In addition to the infinite for loop, this response does not return an output value, which means that even if there is a variable containing a correct calculation, this function does not output this information to the user.
The other participant who wrote an incorrect response using an iterative `for` loop structure and the division operation made an error while truncating the number. Figure 4.11 represents this incorrect solution. This function uses modulus 10 in order to retrieve the least significant digit from the number; however, the participant changed the division each time through the loop. This means that the first time through the loop the function calculates the number modulus 10, which yield the least significant digit, and adds this value to the result. The second time through the loop, the function calculates the original number modulus 100 which yield the one’s and ten’s place of the number and this is added to the result. In addition to adding these values, the function subtracts all the digits besides the most significant each time. For example, if the number is 123, the result will add $3 + 20 + 100$, which will not yield $1+2+3$ as the task indicated. Therefore this solution was considered incorrect.
As described in Chapter 3, a follow up question to this task involved introducing the division operations to participants who had not used it in their original solution. Out of ten participants, seven did not change their solutions, meaning the participants indicated during the interview that they were content with their initial solution that did not use the division operation. Three participants wrote a second solution, using the division operations. Two of the three stuck to their initial iteration loop structure, meaning if their initial solution was a `for` loop, their solution using division operations was still a `for` loop. The other participant switched from a `for` loop structure to a `while` loop structure. All three participants wrote a correct second response using division operations.

### 4.1.3 List Search Task findings.

For the third task, the participants were asked to write a function to return true if a given item is within an unordered list and otherwise
to return false. This task is referred to as the “List Search Task.” First, responses were divided into two groups: solutions to the task that used an iterative approach and ones that used a recursive approach. Horizontal analysis revealed that all seventeen participants, who represent 100% of the participants, used iteration when solving this task and thus none of the participants solved the List Search Task using recursion. Twelve participants chose to solve The List Search Task using a for loop structure, while five participants’ responses did not fit into the for or while loop categories, therefore a category “other” was created and is described below. All of the responses for this task were found to be correct, therefore, all the participants, regardless of the approach used to solve this task, solved it successfully. Figure 4.12 represents the findings from the List Search Task.
4.1.3.1 Iteration: For loop and while loop responses. The second part of the analysis was to divide the iterative responses into for loop and while loop subcategories. None of the participants chose to solve this task using a while loop structure. Twelve participants chose to solve this task using a for loop structure. From the data, a new subcategory “other” also emerged. The “other” subcategory represents participants’ responses that did not fit into the for loop subcategory or the while loop subcategory.

The participants’ solutions that were categorized in the “other” category either gave a verbal response to the task (without including any written work), or included written work in terms of pseudocode. Five participants wrote responses that were
categorized as “other.” Four of these participants did not include written work while solving this task, therefore their verbal solution was considered as their response. The other participant wrote a response using pseudocode notation. Both of the verbal solutions and the pseudocode did not use a for loop structure or a while loop structure but they were all iterative solutions.

4.1.3.2 Correct and incorrect responses. When investigating correct and incorrect responses to the List Search Task, results show that 17 participants, who represent 100% of participants, solved the task correctly and none of the participants solved the task incorrectly. Participants solved this task using different iterative algorithms. Horizontal analysis discovered three different algorithms, which are described below in detail.

One response that was categorized as correct did not include a loop structure, but was in the form of an iterative approach; therefore it was categorized as “other.” The participant did not know how to check the given item against each entity from the list; however, the participant relied on a made-up function called “is in.” The transcript below shows that the participant explained that the function “is in” as an iterative function that iterates through every entity of the list to match the given item.

P2: if I remember this correctly will run through every member, every entity of a list, assuming it is just a list and nothing fancy like that, it will check n against the entity, and if it finds it, it will automatically return true, if it doesn’t it
will return false, I think that’s probably it, I am trying to remember if there is any other details to it.

I: No that’s fine. How do you think is in is running through the list?

P2: How do I think it’s running through there, well, maybe there is other function that return the length of the list, and then it can use some kind of iteration, some iterative for loop, so it can go through each, maybe for loop control variable i and i is less than list dot length or whatever they call that, and then it will go through each one, there is probably more efficient way to do it, but I think it will get the job done.

Twelve participants solved this task correctly with an iterative approach using a for loop structure. Figure 4.13 contains a correct iterative solution using a for loop structure.

```
boolean isLn (int n, List l)
{
    for (i = 0, i < l.size(), i++)
    {
        if (l[i] == n)
            return true;
    }
    return false;
}
```

Figure 4.13. Correct iterative solution to the List Search Task, using a for loop structure.

The second command in the figure is the for loop command. The lower boundary is zero, and the upper boundary is the size of the list. This means that each time through the loop the function will calculate the size of the list. Size of the list operation
can be inefficient if repeated through the loop. This depends on the language – the boundary may not be re-evaluated each time in some languages, and for some implementation of lists, it may not be inefficient to find the length – but in general, it would be considered to as inefficient. When using recursion to solve this task, there is no need to calculate the size of the list. One can repeat the recursive step until the list is empty, without considering the size.

### 4.1.4 Summary of part one results

In conclusion, the horizontal analysis revealed that one participant solved the first task using recursion and 100% of participants solved the second and the third tasks using an iterative approach. The graph in Figure 4.14 shows the numerical values of these results.

![Graph showing results of the first three tasks](image)

Table 4.1. **Results of the** first three tasks.

The aim of this study was to reveal whether students use recursion as a problem solving approach. In particular, the research question was: Do students use recursion
when solving programming tasks? These finding show that student do not use recursion when solving these three particular tasks, the Power Task, Summation of Digits Task and List Search Task. Therefore, it can be concluded that students do not use recursion as a problem-solving technique. Students tend to use iteration when solving these programming tasks.

Research has shown that when given programming problems, which can be solved using recursion and iteration, the majority of students do not use the recursive approach. Instead, they attempt various forward reasoning ways such as iteration, some of which are erroneous and others that are inefficient (Ginat, 2004). The findings of the current investigation are consistent with Ginat (2004) findings.

In addition to investigating when students use recursion, the current study also investigated why students use or do not use recursion as a problem solving approach. Analysis of part one of the interviews shows that students choose to use iteration to solve these three tasks. After establishing that students prefer to use iteration to recursion, the researcher aimed to answer the second research question regarding reasons why students choose to construct an iterative solution over a recursive solution. The next chapter will discuss this.
CHAPTER 5 – STUDENTS SOLVE PROGRAMMING TASKS USING RECURSION

The previous chapter, Chapter 4, presents the findings from the first part of the interview. These results show that students do not use recursion as a tool for solving programming tasks. One of the research questions is: Why do students choose to use recursion or to use iteration when solving algorithmic programming tasks? To make claims about why students choose one approach over the other, first it must be established that the students are able to solve the problems using both approaches. If students are able to solve the problems using both approaches, then the solution they choose is actually a choice and is not just because that is the only way they know how to solve the task. Therefore, it is important to investigate if students who solved the tasks using iteration can actually use recursion as a problem solving approach. In addition, participants were asked to reason about their choice of solution. This chapter presents the findings from the third part of the interview as well as a discussion of answers to the research questions.

5.1 Students can write recursive solutions when prompted: Results from part three of the interview

During part three of the interview, participants were prompted to solve the first three tasks using the solution approach that they had not used during the first part of the interview. The participants’ responses to re-doing the first three tasks were analyzed using the same algorithm as was used with the data from part one of the interview. The
section below contains separate discussions of the findings from data generated by each task. These findings show that when given algorithmic programming tasks, when prompted to use recursion, most participants successfully solve the tasks. Therefore, it can be concluded that most participants can successfully solve these tasks using recursion even though initially they chose to solve them using iteration.

5.1.1 Power Task findings. Recall that 16 participants elected to initially solve the Power Task using iteration and one participant chose to use recursion to solve the task. Therefore, in this part of the interview, 16 participants were asked to solve the Power Task using recursion, and one participant was asked to solve the Power Task using iteration. 11 out of 16 participants, (68.75% of the interviewees) solved the Power Task correctly using recursion. Three participants, who represent 18.75%, solved the Power Task incorrectly using recursion, and two participants, who represent 12.5% of the participants, did not solve the task. The participant who initially solved the task using recursion during part one of the interview, was asked to solve the Power Task using iteration during this part. Analysis shows that this participant solved the Power Task using iteration correctly. Figure 5.1 represents the findings of the Power Task when participants were prompted to use a certain approach to solve the task.
5.1.1.1. Correct responses. Out of 16 participants who were asked to solve the Power Task using recursion, 11 provided a correct response. As described in Chapter 3, data from the first three tasks were analyzed using the same algorithm that was developed by the researcher. Even though 11 participants solved the Power Task correctly, they did not use the same base case. The base case is used in a recursive function in order to stop the recursion. Section 5.1.1.5 presents a discussion of the base case finding in detail, in addition to further explanation of the importance of it.

5.1.1.2. Incorrect responses. Three participants solved the Power Task using recursion incorrectly. One of the participants provided an incorrect recursive response to the Power Task that involved overwriting the variable x. In other words, each time
through the recursive call of the function, the participant over wrote the value of \( x \) by multiplying it by itself and storing the result as \( x \). In particular, the first time though the function, \( x \) is equal to 4, however, \( x \) gets the multiplication of itself as a new value, therefore \( x \) is equal to \( 4 \cdot 4 \), which is 16. During the second time through the function \( x \) is equal to 16, and it will be multiplied by itself, which will yield \( 16 \cdot 16 \), and stored as the new \( x \). Therefore, now \( x \) is equal to 256. The function will keep excising until the exponent is zero. In addition to incorrect calculation of exponent, this participant did not write a return statement within the recursive function. Without a return statement the function will not build up a solution. This mean that the element of passive flow in missing in this solution. Figure 5.2 contains an incorrect recursive solution that involves overwriting the variable \( x \). The second line represents the overwriting of \( x \) as \( x \) times itself.

![Figure 5.2. An incorrect recursive solution to the Power Task](image.png)
Out of the three participants who provided an incorrect response, one participant incorrectly used the addition operation in order to calculate the power of $x^n$. Instead of multiplying the result by $x$, the participant added the outcome of the multiplication to the variable result. In particular, the participant initialized result to one, then multiplied result by the base of the exponent, $x$, which yield 4, and then added this calculation to result. By the end of the first execution, result was equal to $4 + 1$, which is 5. The second time through the function, result was again multiplied by 4 and added to result. By the end of the second time though, the recursive function result was equal to 25. Because this response does not yield a correct calculation of $x^n$, this response was categorized as incorrect. Figure 5.3 contains an incorrect recursive solution that involves overwriting the variable result.

![Recursive Solution](image)

Figure 5.3. An incorrect recursive solution to the Power Task

The last participant who solved the Power Task using recursion incorrectly wrote an infinite recursion function that did not yield a correct calculation of $x^n$. The participant
did not include a base case, therefore the function will execute infinitely many times. Even though the participant wrote the recursive call on a reduced input, when the base case does not exist, the recursion will not stop. Figure 5.4 contains an incorrect recursive solution that lacks a base case.

![Figure 5.4. An incorrect recursive solution to the Power Task](image)

5.1.1.3 Did not solve the task responses. Two participants did not attempt to solve the Power Task using recursion. One participant stated that “Yeah. I could solve these tasks using recursion, but I don’t want to.” Since the participant refused to solve the first three tasks when prompted to use recursion, the response was categorized as “did not solve.” The other participant who did not attempt to solve the Power Task using recursion was not comfortable with writing a recursive solution:

P: So, this one we are just going to say exp, say int x int n and it is going to do an if umm, [Pause for 16 seconds] Maybe I don’t, I am trying to kind of look at that one, umm... So, do I need to like do I need to have a counter? That like decrement or increment? Yeah well looking at this example I guess I kind of understand but I don’t, like I don’t understand how it goes through and like actually decrement n.
5.1.4 Correct iterative solution. Out of 17 participants, only one participant chose to initially solve the Power Task using recursion during part one of the interview. Therefore, only this particular participant was asked to solve the Power Task using iteration during the third part of the interview. This participant solved the Power Task using for loop structure. Figure 5.5 contains the response that was categorized as a correct iterative response using for loop.

```
def power(x, n):
    prod = x
    for i in range(1, n):
        prod *= x
    return prod
```

Figure 5.5. A correct iterative solution to the Power Task using a for loop

5.1.5 Base case findings. When writing a recursive function, one must write a base case for the recursion function in order to halt the execution. Research has shown that students have difficulty with recursion, in particular regarding the base cases (Haberman & Averbuch 2002). The base case of each task is described below. This is stated because part of the analysis focused on whether participants of this study had difficulties with base case.

Out of 16 participants who were asked to solve the Power Task using recursion, 11 provided a correct response. Among these correct responses, participants used different base cases in order to stop the recursion. Sub-categories related to these
different base cases emerged from the analysis of the data. When solving the Power Task using recursion, one can stop the recursion when the exponent is equal to zero, calculating the case of \(x^0\), which will return 1. Another correct base case would be when the exponent is equal to one, calculating the case of \(x^1\), which will return \(x\). The base case \(x^0\) will require one more recursive call over the base case of \(x^1\). When returning the base case, the result will be multiplied by the previous function call; therefore, there is no benefit or drawback regarding the result of the function when reaching the base case of \(x^0\), which will multiply the previous function call by 1.

Eight out of 16 participants who solved the Power Task correctly using recursion used the base case of \(x^0\). Two participants solved the Power Task correctly using the other base case of \(x^1\). One of the participants solve the task in such that the recursion will halt when \(n\) is less then 2, which is mathematically equivalent to when \(n\) is equal to 1. Therefore it can be concluded that eight participants used the base case of \(x^0\), and three participants used the base case of \(x^1\). This study aims to discover if students use recursion as an approach to solve algorithmic tasks. If students are having difficulties with recursion, in particular with the base case, it can be claimed that this will prevent them from using recursion as an approach to solve tasks.

Two participants solved the Power Task using recursion incorrectly. Both participants used a correct base case of \(x^0\). Both participants added the value of \(x\) to the recursive call, which will not yield a result of \(x^n\). Even though the base case was correct, the calculations of \(x^n\) were incorrect; therefore the responses were categorized as incorrect.
To summarize, 16 participants were asked to solve the Power Task using recursion and one participant was asked to solve it using iteration. When solving the Power Task recursively, eleven participants solve it correctly, three solve it incorrectly and two participants did not attempt to solve the Power Task.

5.1.2 Summation of Digits Task findings. Since all the participants solved the Summation of Digits Task using iteration during the first part of the interview, during the third part of the interview they were all asked to solve it using recursion. 11 out of 17 participants, who represent 64.7%, solved the Summation of Digits Task correctly using recursion. Two participants, who represent 11.77%, solved the Summation of Digits Task incorrectly using recursion, and four participants, who represent 23.6% of the participants, did not attempt to solve the task. Figure 5.6 represents the findings of the Summation of Digits Task when participants were prompted to use recursion in order to solve the task.
5.1.2.1. Incorrect responses. Two participants solved the Summation of Digits Task using recursion incorrectly. One of the participant’s response still included iteration as an approach to break the number down to its digits. Since the participants were asked to use recursion to solve the task, the fact that this participant used iteration again meant this response was categorized as incorrect. In addition to using iteration, the participant wrote an infinite recursion function. Instead of calling the function with a smaller input, the participant called the function with the same domain size, which will cause an infinite recursion. Figure 5.7 contains a participant response that was categorized as an incorrect recursive solution to the Summation of Digits Task. The third statement contains the iterative step to break down the number to its digits. The fifth statement line contains the
recursive called on the variable $a$ without a change which means the domain is not changing.

The second participant who provided an incorrect recursive response made an error in the base case. The participant separated the least significant digit from the number using modulus, and stored it in a variable called Mod. The base case was an if statement (condition) asking if the variable mod is less than 10. This statement will always be true, since mod is a single digit. Therefore, the function will stop and will not continue to the recursive step. This will not calculate the Summation of Digits Task of a number, therefore this response was categorized as incorrect. Figure 5.7 contains a participant response that was categorized as an incorrect recursive solution to the Summation of Digits Task.
5.1.2.2 Did not attempt responses. Four participants did not attempt to solve the Summation of Digits Task using recursion. One of the participants stated, “it would take a little while because I don’t do recursion all that often, I mostly dealt with iteration. I haven’t done too much that required recursion.” This participant did not attempt to solve the Summation of Digits Task but correctly solved the previous task using recursion. Two of participants who did not attempt this task also did not attempt the Power Task when asked to solve it using recursion. Another participant who did not attempt to solve this task solved the previous task using iteration incorrectly.

5.1.2.3 Base case findings. Out of 17 participants, 11 solved the task correctly using recursion. Within the correct responses, participants used different base cases in order to stop the recursion. These different base cases produced different categories of base cases that emerged from the data. When solving the Summation of Digits Task using recursion, one can stop the recursion when the number is equal to zero. This means all the least significant digits were already pulled out and the number is left as zero or another correct base case can be when the number divided by 10 is equal to zero, meaning the

```
def summation(number):
    mod = number % 10
    if (mod == 0):
        return mod
    else:
        number = (number / 10)
    return mod + summation(number)
```
number only includes one digit. Another way to stop the recursion is to ask when the number is less than 10, which might be more efficient than to call the mod function again. The drawback to using one base case over the other is the number of times the recursive call is executed. The result of the base case will be added to the previous function call.

Eight out of 11 participants who solved the Summation of Digits Task correctly using recursion used the base case of number is equal to zero. Three participants solved the Summation of Digits Task correctly using the other base case of number divided by 10 equal to zero.

To summarize, 17 participants were asked to solve the Summation of Digits Task using recursion. When solving the Power Task recursively, 11 participants solved it correctly, two solved it incorrectly and four participants did not attempt to solve it.

### 5.1.3 List Search Task findings

Because all the participants solved the List Search Task using iteration during the first part of the interview, during the third part of the interview they were asked to solve it using recursion. Ten out of 17 participants, who represent 58.8%, solved the List Search Task correctly using recursion. One participant, who represents 5.8%, solved the List Search Task incorrectly using recursion, and six participants, who represent 35.6% of the participants, did not attempt to solve the task. Figure 5.9 shows the findings of the List Search Task when participants were prompted to use recursion in order to solve the task.
5.1.3.1. Incorrect responses. One participant provided an incorrect recursive response to the List Search Task that involved trying to write the binary search solution. The participant assumed that the list was sorted and therefore a binary search would work, however, the list in the task is not sorted and therefore dividing the list in the middle and searching will not yield a correct response. In addition to implementing binary search on an unsorted list, the participant made an error within the nested if statement. The second if statement compares the middle entity of the list with the given item in order to check if the middle entity of the list is greater then the item given. If this condition returns true and the list is sorted, the recursive call should only investigate the entities above the middle position. Otherwise, the if condition is false, which means that the middle item of the list is smaller than or equal to the given item. There should be
another if statement to ask if the middle entity of the list is smaller than the item given but this is absent in the response. The reason to separate the two different cases (the case where the middle of the list is smaller than the given item and the case where the item given is equal to the middle of the list) is due to different actions following the different base cases. In other words, for one base case, one statement will be executed, and for the other base case a different statement will be executed. If the item in the middle of the list is smaller, then the function should use recursion to check the lower part of the list with the given item. If it is the case where the middle item of the list is equal to the item given, the function should return true. An if condition is required in order to separate the two base cases. In this response, this if condition is missing, therefore, the solution presented in Figure 5.10 would keep truncating the list without the lower end. This will not progress to execute the comparison between the entity and the given item, therefore the function will never return true. This was considered an incorrect response.
5.1.3.2 Did not attempt responses. Six participants did not attempt to solve the List Search Task when asked to solve it using recursion. Two of participants who did not attempt this task also did not attempt the previous two tasks (Power Task and Summation of Digits Task) using recursion. Two other participants who did not attempt to solve this task solved the Power Task using iteration incorrectly and did not attempt to solve the Summation of Digits Task. One participant, who did not attempt to solve this task, solved the Power Task correctly using recursion and did not attempt the other two tasks. And the last participant solved the Power Task correctly using recursion, the Summation of Digits Task incorrectly using recursion, and did not attempt to solve the List Search Task. Participants were not asked to verbalized why they did not attempted to List Search Task; however, 4 out of 6 participants did not solve the Summation of Digits Task and the
Power task, which suggests that these participants are not comfortable with constructing recursive solutions. The other two participants had trouble identifying the two different base cases, and after some time the participants chose to move forward to the next part of the interview.

**5.1.3.3 Base case findings.** When solving the List Search Task using recursion, two different base cases must be used in the same function. The first one is when the given number is found to be equal to the number at the first position of the list thus the function should return true. The second base case is when the number is not found, which means that the function should return false. Meaning, the recursive function cannot examine any more items because we are either at the end of the list or the list is empty. These two base cases must be presented in the solution in order to yield a correct response. Out of 17 participants, 10 participants solved the task using recursion and did so correctly. Within the correct responses different base case groups emerged from the data. For example, one participant wrote only one base case condition checking if the item was in the list, without returning the false statement. This response was categorized as correct because the searching of the item within the list was written successfully with a minor error of missing statement to return false if the item is not in the list.

**5.1.3.4 Sorted List Search findings.** As described in Chapter 3, participants were asked a follow up question to the List Search Task. They were asked whether their responses would change if the task asked them to search an item in a sorted list. The goal for searching an item in an ordered list could trigger participants to consider a different
solution. Therefore, the goal was to investigate if an ordered list would yield a recursive approach when an ordered list had yielded an iterative approach.

Ten participants indicated that if the task had asked them to search for an item in an ordered list they would have used Binary Search algorithm in order to solve the task. When asked how Binary Search would be implemented, seven participants responded that Binary Search is implemented using recursion. Three participants, however, believed that Binary Search is implemented using an iterative loop. Binary search can be written as an iterative solution, however it is more complicated and longer than the recursive solution. This shows that tasks involving an ordered list and unordered list could yield different approaches from participants when they are asked to solve the searching task.

**5.1.4 Summary of Part three.** In conclusion, analysis of part three of the interview shows that 64% of the responses were categorized as correct, 12% of the responses were categorized as incorrect and 24% responses was categorized as did not attempt. Table 5.1 represents the results of how successful participants were who were asked to solve the task recursively during this portion of the interview. The left column of each task represents how many participants successfully solve the task using recursion. The middle column of each task shows how many participants were unsuccessful when solving the task using recursion, and the right column of each task represents how many participants did not attempted the task when prompted to use recursion.
Table 5.1 Accuracies of participants for each task using recursion

Table 5.2 shows the number of participants who successfully solve the task using recursion, were unsuccessfully in solving the task using recursion and the number of participants who did not attempt to solve the task when prompted to use recursion. Since most participants solve the task correctly when prompted to use recursion, it can be concluded that students are able to solve the task using recursion; however, recursion is not their chosen approach.

<table>
<thead>
<tr>
<th>Task</th>
<th>Power Task</th>
<th>Summation of Digits Task</th>
<th>List Search Task</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>Incorrect</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Did not attempt</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 5.2 Frequency of participants’ solution accuracies using recursion for each task
The next section describes various reasons that participants gave when asked why they choose to solve these tasks using iteration even though they were successful solve these tasks using recursion.

5.3 Why students initially solved the tasks using iteration

As described in Chapter 3, after participants solved the first three tasks, they were prompted to solve the tasks using the other approach. They were also asked why they initially solved the tasks using the approach they had chosen. Analysis of these responses generated four main response categories: “stack overflow,” “properties of recursion,” “curriculum related or teaching related,” and “other.” Table 4.4 presents a complete listing of responses categories with the frequency with which each category appeared.

<table>
<thead>
<tr>
<th>Response category</th>
<th>Number of participants Providing Responses Coded as Given Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Overflow</td>
<td>4</td>
</tr>
<tr>
<td>Properties of Recursion</td>
<td>3</td>
</tr>
<tr>
<td>Curriculum related or teaching related</td>
<td>8</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.3 Response categories to why solving the tasks iteratively

5.3.1 Stack overflow category. The first category is “stack overflow.” Stack overflow is a situation that occurs when too much memory is used on the call stack. The call stack, or just stack, is a portion of memory that is used to hold local data and other
information about function calls. When a function is called, including a recursive function, its return address, parameters, and local variables are “pushed” onto the stack as a stack frame; when it is done, its stack frame is “popped” off the stack and so discarded. The call stack contains a limited amount of memory, often determined at the start of the program. The size of the call stack depends on many factors, including the programming language, machine architecture, multi-threading, and amount of available memory. When a program requires more space than is available on the call stack, the stack is said to overflow, typically resulting in a program crash. An infinite recursion or a very deep recursion can often cause stack overflow. Some participants were very explicit about stack overflow and chose the iterative solution in order to avoid such situations from possibly occurring. For example, one participant said, “Because for, like in this case [the Power Task] for very large values of \( n \) you can overflow the stack, and because for each recursive call you have to create a new stack frame, you know... You drop down into that.”

When asked why they initially chose to solve the tasks using iteration, four out of 17 participants gave a verbal response that included a discussion about stack overflow. Some participants were very explicit about not using recursion in order to avoid stack overflow and chose to solve the tasks with an iterative approach. Another participant said:

For little things you really don’t want to do recursion because, ‘cause every time the recursive call, it has to make like a new stack, it pushed everything it has on to the stack and change it so that new function call. So if you keep doing that over and over, it kind of going to the stack overflow.
5.3.2 Properties of recursion category. The second category is called “properties of recursion.” When asked why their initial solutions used an iterative approach, three out of 17 participants discussed properties of the recursive solution. Even though a recursive solution might be shorter than the iterative solution, more readable and look simple and elegant, these participants reasoned that a recursive solution is complicated to write, takes more time and it is hard to visualize the flow of the recursive function. One participant stated:

Though I like recursion... If I already have the solution done, but at first glance like you saying loops, I do not why I avoid recursion, I guess I just... it takes a while for me... It’s a lot harder to visualize I guess because you are going down through this recursion tree, or what not and it’s really hard to like... when you have numbers like 10 if you decrementing by 1 to start at those are then recurse through them in your head but with a loop its a lot easier to see, I guess.

Another participant stated that:

It is probably the same number of lines of code but recursion is a concept that is somehow more complicated [and it] something that students have trouble with, when they first learning computer science so, it... So I want to say that I was avoiding recursion, cause it look like it would complicated it but really the recursive solution looks simple to me as non recursion solution so...
5.3.3 Curriculum related or teaching related category. The third category is called “curriculum related or teaching related.” This is the predominant response category because eight out of 17 participants gave a verbal response including a discussion about the curriculum they had experienced in computer science courses. Participants whose responses fit in this category believe that because they learned iteration first in their programming or computer science classes, they choose to turn to iteration rather than recursion as their favorite approach to solve programming tasks. One participant mentioned that he or she does not use recursion and turns to iteration as the initial solution “because that is what I was first introduced to. In programming books. Yeah, they don’t really throw recursion in until like, eight chapters in.” Another participant stated, ”when you take like programming, like classes with a lot of programming, and umm, I mean like basic programming classes, they always treat recursion as some sort of like advance topic, you first learn a lot of iteration like for loops and while loops.” It is worth noting that all the participants in this study were taught iteration prior to recursion in their first programming course.

5.3.4 “Other” category. The fourth category is called “other.” Two participants’ responses did not fit into any of the other categories mentioned previously and therefore were categorized as “other.” One participant mentioned that they do not think recursively and said:

Would try to think of things more recursively if I knew that coding it that way was always the going to be better choice, I don’t feel like it is really to my
advantage to think of it that way, when I won’t code it that way. So I don’t think I would have put an effort into thinking that way.

Another participant mentioned:

I just see things in terms of lists, like well a lot of things I can just modeled in terms of list and it makes it easier for me cause I can... because when I did the iteration I turned that into a list of integers anyway so I just saw looping through a list.

5.4 Base case discussion

When writing a recursive function, one must write a base case for the recursion in order to halt the execution of the function. Research has shown that students have difficulties with recursion, in particular regarding the base cases of recursive functions (Haberman & Averbuch 2002). Haberman and Averbuch claim that these difficulties with the base case might affect how students perceive the correctness, readability, and code complexity of recursive functions. Even though they claim that students have difficulties with the base case, when students write recursive functions, eventually students have to think of how to split the problem into sub-problems, in addition to one or more base cases. Students, who have difficulties identifying the base case usually either use redundant base cases or they may ignore the procedural aspects of the recursive process, and therefore avoid essential stopping condition. When one is avoiding the essential stopping condition, one creates an infinite recursion.

Previously in this chapter the various base cases of each task were identified and discussed. Most participants did not present key difficulties identifying the base case of
the task. When solving the Power Task recursively, one participant did not write the base
case of the function, meaning he or she created an infinite recursive function, which was
categorized as incorrect response to the task. When solving the Summation of Digits Task
recursively, none of the participants made an error in the base case part of the recursive
response. Three participants did not attempt the Summation of Digits Task, and it could
be that because they did not identify the base case, they choose to not attempt the task,
but there is no evidence to support this claim. Analysis of the List Search Task
recursively shows that six participants did not attempt to solve the task. As discussed in
4.3, List Search Task required two different base cases combined, which means this task
might be more difficult than the Power Task and the Summation of Digits Task. It can be
speculated that the participants did not attempt the task because they failed to identify the
two different base cases for the List Search Task, but there is no specific evidence to
support this claim.

In summary, even though participants did not appear to have major difficulties
with the base case, it may be that the identification of the base case caused difficulties for
participants who did not attempt to solve the interview tasks when prompted to use
recursion. However, there is no conclusive evidence to support this claim.

5.5 Reasons for solving the tasks using iteration - discussion

After participants solved the first three tasks using the other approach, they were
also asked why they initially solved the tasks using the approach they had initially
chosen. Response categories were described previously in this chapter. Presented below
is discussion of each category.
5.5.1 Reasons for solving the tasks using iteration: Stack overflow. Four participants said that the reason they initially solve the tasks iteratively is due to stack overflow. Stack overflow is a situation that can occur when implementing a recursive solution. Some participants said that they chose to solve the tasks using an iterative solution in order to avoid such situation from possibly occurring. However, stack overflow is not a valid reason due to the fact that all the tasks presented in this study are simple programming tasks that do not cause stack overflow. For example, when writing a recursive factorial solution and testing the code using 50!, one might believe that stack overflow error should occur. However, the error the programmer will receive would be about the length of the number. The system cannot allocate a large variable to hold all the individual digits for 50! Stack overflow could occur in an infinite recursion situation, since the recursion will never stop, at some point the depth of the stack will be finite, and a stack overflow error will be prompted to the user.

Difficulties with the concept of stack overflow is not well documented in the research; however, Regehr at el. (2005), developed a practical static analysis that showed a solution to a certain task can avoid stack overflow and can provide accurate results. This shows that stack overflow may not necessary occur when implementing and executing a recursive solution, therefore not choosing a recursive solution due to stack overflow it not an appropriate reason.

For the present study, the researcher also made trail runs, with both factorial and binary tree tasks, using the iterative solution and the recursive solution. For big numbers such as 50! and a binary tree depth of 50 levels, the recursive algorithm yields a valid
result and does not halt due to stack overflow. This shows that the stack overflow error that participants discussed was not likely to occur using the algorithm presented in the interview.

5.5.2 Reasons for solving the tasks using iteration: Recursion properties.

Three participants included a verbal response that was categorized as “properties of recursion.” Recursive solutions may be more difficult to design and test, however the solutions are often shorter and closer to the abstract mathematical nature of the task. Since the recursive solution is shorter, it may look more appealing, elegant and simple. Participants’ responses that were categorized as recursion properties category discussed that recursion can be very difficult to read and to write. Recursion is an abstract concept and in order to construct a recursive solution one must identify the three core elements of recursion: the active flow (or the recursive call), the base case and the passive flow (the returned values). Since the passive flow occurs when all the active flows are done executing, it can be said that the passive flow is an abstract element of recursion that can make it difficult to write. All three element of recursion can be considered a property of a recursive solution, even though it might be harder to envision, and difficult to identify. However, recursion is usually a very short, simple looking solution, therefore it appearance is a positive property of recursion.

Participants mentioned that they did not chose to solve the tasks initially using recursion due to the fact that it was difficult and can be hard to visualize. Research supports the fact that recursion is an abstract concept (Feinberg, 2007; Kurtz & Johnson, 1985; Stephenson, 2009; Stern & Naish, 2002), and it might be hard to visualize the
passive flow of the recursive function (Uri Leron, 1988). Leron (1988) claims that “both the power and the complexity of recursion stem from the same source: its ability to yield very simple descriptions to highly complex phenomena” (pg 1).

Students verbalized that building up a result from all the recursive calls combined is difficult to envision; therefore, they consider the recursive solution difficult to construct.

5.5.3 Reasons for solving the tasks using iteration: Class related. As mentioned previously, findings show that participants do not choose to use recursion when asked to solve algorithmic programming tasks. After solving the first three tasks (the Power Task, the Summation of Digits Task task, and the List Search Task) using both approaches, the researcher asked each participant to discuss the main reason why he or she initially solved the task using the certain approach they used. The current chapter presented the complete listing of responses categories with the frequency with which each category appeared. Four categories were identified: “stack overflow,” “properties of recursion,” “class or teaching related reason,” and other. Two out of the four categories presented, “stack overflow” and the “properties of recursion,” were discussed above. Eight participants’ responses were categorized as “class or teaching related reason.” Participants’ responses that were categorized in this category presented the reason related to computer science curriculum. Participants’ stated that when they learn to program, the first concept taught was iterative loops, such as for loops and while loops and only later on during the course the concept of recursion was taught. In particular, participants believe that because iteration was taught first, this is the approach they choose to use. In
addition, some participants mentioned that they had more practice using iterative loop, due to the fact that those were taught first.

Research has shown that that teaching recursion before iteration can lead to deeper learning of programming and students tend to leave the first programming course with better problems solving skills than in the previous incarnation (Mirolo, 2010; Turbak & Royden, 199AD; Wiedenbeck, 1988). However, participants in this study were taught iteration prior to recursion, which might lead them to believe that iteration is the approach they are more apt to turn to when solving algorithmic programming tasks. Students in this study learn iteration first prior to learning and practicing recursion. It can be inferred that because students who learn iteration prior to recursion are not turning to recursion due to lack of deeper understanding of recursion. However, findings from this study show that most of the participants were successful in solving programming problems using recursion, even though they learned iteration prior to learning recursion. Therefore it can be concluded that students can solve programming tasks using recursion when prompted to do so even if they would prefer to use iteration instead.

5.6 Summary

Findings shows that during the first part of the interview participants solved the three tasks using iteration, while findings from part three show that participants can successfully solve these tasks using recursion. To answer the researchable question, participants do not choose to use recursion when ask to solve algorithmic programming tasks. However, even though students are able to use recursion as a tool to solve programming tasks, participants do not choose recursion. Participants mostly used
iteration as their initial approach to solve the interview tasks. Participants were asked why do they think they used recursion and analysis generated four main response categories: “stack overflow,” “properties of recursion,” “curriculum related or teaching related,” and “other”. Stack overflow, can occurs when a program requires more space than is available on the call stack, the stack is said to overflow, typically resulting in a program crash. An infinite recursion or a very deep recursion can often cause stack overflow. Participants who discussed the simplicity and elegance of the recursive solution, or discussed the difficulties in visualizing the result as it is built up from the recursive calls were categorized as properties of recursion. Participants who reasoned using iteration because they have been taught iteration prior to recursion were categorized as curriculum related or teaching related reasons. To summarize, participants presented different reasons to the question why did they choose iteration as their initial solution to the first three tasks.
CHAPTER 6 - STUDENTS’ CHOOSING RECURSIVE SOLUTION

As described in previous chapters, our results show that students do not use recursion as a tool for solving programming tasks even though students can successfully solve programming tasks using recursion when prompted to do so. During the interview participants were also asked to choose between an iterative and a recursive solution to a certain task. In addition, participants were asked how they initially defined the tasks. This chapter presents the findings from these parts of the interview and a discussion of answers to the following researchable questions: when presented with an iterative solution and a recursive solution, do students choose one solution over the other? Why do they choose the recursive solution? Why do they choose the iterative solution? How do students define the problem as a set of sub-problems? Do they define the objects of the problems as recursive objects?

6.1 Students prefer the recursive solution when given a choice between two different solutions: Results of part two of the interview

During part two of the interview, participants were asked to pick solutions to two different tasks, each of which was presented along with two possible solutions. One solution was an iterative solution and the other was a recursive solution. As described in Chapter 3, data from these two tasks were analyzed using an algorithm that was developed by the researcher. The algorithm included two steps. During the interview participants were given two solutions, an iterative solutions and a recursive solution. Participants were ask to verbalize: ”which solution would you choose and why?” The first step was to quantify how many participants choose the iterative solution and how
many participants choose the recursive solution. The second step was to categorize the verbal responses of the participants when they were asked why they chose that solution.

The findings show that when given programming tasks with two different possible solutions, one iterative and one recursive, participants are apt to choose the recursive solution over the iterative solution. Data show that 11 out of 17 participants chose the recursive solution to the factorial task and 13 out of 17 participants chose the recursive solution to the binary tree task. In both cases, more than 50% of the participants chose the recursive solution over the iterative solution. The section below contains separate discussions of the findings from each task and a summary of the results.

6.1.1 Factorial task findings. Participants were asked to choose between two different solutions to the factorial task. The task was to calculate n! (factorial) given a positive integer n. Participants had to choose between a correct iterative solution and a correct recursive solution. The task was to write a function that will compute n factorial. After stating the task, participants were asked to consider the following solutions, and choose the one that they prefer. Unlike part one and three of the interview, there was no need to group the data in correct and incorrect categories. Correct and incorrect categories are needed when participants may successfully or unsuccessfully solve a certain task. In this part of the interview, participants had to choose a solution. However, both solutions presented to participants were correct and solved the task successfully. The data were divided into two different categories: an iterative category and a recursive category. Each category included the number of participants who chose the certain solution.
The horizontal analysis shows that 11 out of 17 participants, who represent 65% of the participants, chose the recursive solution. 35% of participants, represented by six out of 17 participants, chose the iterative solution. This suggests that participants are more apt to choose a recursive solution when presented with two different solutions to solve the factorial task.

6.1.2 Response categories. After participants chose their solution, they were asked to justify why they picked one solution over the other. The researcher identified keywords from the transcripts and the audio files of the interview. These keywords are discussed in each category below.

After analysis was complete, four dominant response categories were apparent: “stack overflow,” “properties of recursion,” “lack of confidence in solution,” and “other.” Table 4.1 presents a complete listing of responses categories with the frequency with which each category appeared.

<table>
<thead>
<tr>
<th>Response category</th>
<th>Number of participants Providing Responses Coded as Given Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack overflow</td>
<td>3</td>
</tr>
<tr>
<td>Properties of recursion</td>
<td>11</td>
</tr>
<tr>
<td>Lack of confidence in solution</td>
<td>2</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1. Response categories why students chose one solution over another when considering the factorial task
6.1.2.1 **Stack overflow category.** The first category is called “stack overflow.”

Three participants, who represent 17.6% of the participants, gave a verbal response that included a discussion about stack overflow. Discussion of stack overflow was verbalized when rejecting the recursive solution and preferring the iterative solution. When a participant mentioned the phrase, “stack overflow” or “it might overflow the stack,” the response was categorized as stack overflow. These two phrases were considered to be keywords that classified the response as being in the stack overflow category.

Stack overflow is a situation that occurs when too much memory is used on the call stack. The call stack contains a limited amount of memory, often determined at the start of the program. The size of the call stack depends on many factors, including the programming language, machine architecture, multi-threading, and amount of available memory. When a program attempts to use more space than is available on the call stack, the stack is said to overflow, typically resulting in a program crash. An infinite recursion or a very deep recursion can often cause stack overflow. Some participants were very explicit about stack overflow and chose the iterative solution in order to avoid such situations from possibly occurring. For example, one participant stated:

So they both solve the problem, but this [recursive solution] one involves multiple recursive calls, so you just keep burning down to the stack and this one [recursive solution] is susceptible to stack overflow, whereas the one on the left should not be, I mean you could in theory, but it’s not the same kind of kind of recursive call structure that you run into.
This participant’s response (and similar ones given by other participants) was coded as being in the “stack overflow“ reasoning category and his/her choice of solution was the iterative solution.

6.1.2.2 Properties of recursion category. The second category is called “properties of recursion.” This is the predominant response category where 11 out of 17 participants gave a response involving a discussion regarding one or more properties of the recursive solution. Every recursive solution can be written iteratively and vice versa. Recursive solutions may be more difficult to design and test, but the solutions are often shorter and closer to the abstract mathematical nature of the task. Since the recursive solution is shorter, it may look more appealing, elegant and simple. Simplicity, elegance and readability are all properties of a recursive solution.

All participants whose responses were in this category chose the recursive solution as their preferred solution to the factorial task. When a participant mentioned that “the recursive solution look simpler,” or “this [solution] is more elegant,” or “the recursive solution seems easier” these were identify by the researcher to be keywords. These keywords were categorized as properties of recursion reasoning.

In responses in the property of recursion category, participants discussed the length, simplicity and readability of the code. Some participants discussed the simplicity of the code: ”If I were to write it I would do it that way [recursive way] it is really simple to write and really easy to think about.” Some participants discussed the length of the code: “Umm, I don’t know, code usually is fairly long, I usually kind of compress it a bit. Trying to like save space that way... Depending on... Like depends... you know code
Other participants discussed the simplicity and elegance of the recursive solution:

I probably chose the one on the right, because it uses recursion, and I don’t know I guess I feel that recursion.... You can do any recursion with loop and you can do any loop with recursion but I think that recursion is a lot.... Looks a lot better, less, its less amount of code, so I guess that’s kind of better.

Another property of the recursive solution is readability. Some participants chose the recursive solution because they believe it is more readable, and easy to follow: “I find the readability [of the recursive solution] to be easier.”

6.1.2.3 Lack of confidence in a solution category. The fourth category is called “lack of confidence in solution.” The researcher informed the participants during the interview that both solutions, the iterative one and the recursive one, were correct. However, some of the participants wanted to check the correctness of the solution, and in doing so, lost confidence in a certain solution and chose the other solution. Keywords that were identify by the researcher included “I am not sure how this works” or “I am not familiar with recursion.” These keywords helped categorized the response as lack of confidence in the solution.

Two out of 17 participants gave a verbal response that included a discussion about their lack of confidence about a certain solution. The lack of confidence with a certain solution was the sole reason why they decided to go with the alternative choice of solution. One of the participants chose the iterative solution and said that: “I don’t get how this [the recursive solution] works… Yeah I won’t be convinced until I can compile
it, so I am going to go with the other one…” Another participant reasoned that he or she chose the iterative solution because he or she was not familiar with recursive solutions: “I guess I will probably go with this one [points to the iterative solution] cause umm, the one on the left, because the one on the right, umm is recursive and I am not really familiar with recursive, or I am not good with recursive functions.” Both participants whose responses were grouped in this category chose the iterative solution as the solution to the factorial task. The reasons given by both participants were a lack of confidence that the recursive solution did indeed solve the task.

6.1.2.4 “other” category. The fourth category is called “other.” One participant’s responses did not fit into any of the other categories and therefore was categorized as “other.” In this response, the participant showed knowledge that the recursive solution might be causing issues, but he or she still chose the recursive solution. He or she thought of the factorial task as a recursive task and therefore the recursive solution seemed more intuitive:

P: [Pause for 12 seconds] I think I would have written this one [points to the recursive solution] but it would be bad for really large n. [Circles the recursive solution.]

I: And why do you think you would write the recursive one?

P: Factorial is more intuitive; it’s natural to think of it that way.

6.1.3 Binary tree task findings. Another task used during the second part of the interview was the Binary Tree task. This task asks the participants to choose a solution to
the task of counting the nodes in a binary tree. The horizontal analysis shows that 13 out of 17 participants, who represent 76.4% of the participants, chose the recursive solution. 23.6% of participants, represented by four out of 17 participants, chose the iterative solution.

**6.1.4 Response categories.** After participants chose a solution, they were asked why they chose this solution over the other. After analysis was complete, five response categories were apparent, four of which are equivalent to those developed from the factorial task data. One additional category called “structure resemblance” emerged from the data. Table 4.2 presents a complete listing of responses categories with the frequency with which each category appeared.

<table>
<thead>
<tr>
<th>Response category</th>
<th>Number of participants Providing Responses Coded as Given Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack overflow</td>
<td>3</td>
</tr>
<tr>
<td>Properties of recursion</td>
<td>8</td>
</tr>
<tr>
<td>Lack of confidence in solution</td>
<td>3</td>
</tr>
<tr>
<td>Structure resemblance</td>
<td>2</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.2. Response categories why students chose the recursive solution when considering the binary tree task
Earlier in this chapter the first four response categories were defined and examples were included. Here the findings from Binary Tree task are described. Three participants discussed the possibility of stack overflow, which can be a downfall of using recursion. Eight participants discussed the length, simplicity and readability of the recursion solution and their responses were categorized as “Properties of recursion.” Even though both the iterative solution and the recursive solution were correct, three participants were not certain that a solution solved the task and therefore chose the other solution. These responses were categorized as “lack of confidence in solution.” One participant’s response did not fit into any of the first three categories and therefore was categorized as “other.” The fifth category is discussed below.

6.1.4.1 Structure resemblance category. As mentioned above, an additional (fifth) category emerged from the data. Two participants’ responses included discussion of the structure of the task. Figure 6.1 shows that a binary tree can be represented with a recursive structure. A binary tree consists of multiple sub-trees that each have a similar structure to the binary tree.

Similarly to the recursive solution, where the solution to the problem is divided by combining all the solutions obtained by the sub-problems, binary tree is a structure that has a similar repeated sub-structures. In particular, participants chose the recursive solution because they considered the binary tree to be a recursive structure. One of the participants said, “I would [chose recursion]. Especially because, it’s a recursive structure.” The other participant mentioned, “Recursion seems to work out nicely,
especially ‘cause I guess a tree is like a natural recursive structure that the recursive function seems to make itself for it.”

![Binary tree diagram](image)

**Figure 6.1 Example of a binary tree as a recursive structure**

### 6.2 Participants chose the recursive solution when presented with two different solutions.

Data analysis shows that participants chose the recursive solution when two solutions were presented. Findings from the first part of the interview show that participants choose not to solve algorithmic tasks using recursion. However, participants do choose the recursive solution over the iterative solution when given the choice between the two solution types. Thirteen participants chose to solve the first three tasks using iteration, however, these participants chose the recursive solution over the iterative solution for both the factorial and the binary tree tasks.
Recall that only one participant chose to solve the power task using recursion. During the second part of the interview, however, this participant chose the iterative model for the power task, even though he or she had solved it using recursion. For the factorial task and binary tree task this participant chose the iterative solutions and justified those choices because of wanting to avoid stack overflow.

One goal of this study was to investigate whether students solve programming task using recursion spontaneously (e.g. without being prompted to use that method). Findings from the first part of the interview show that participants did not attempt to solve the interview tasks using recursion. Instead, participants solved the tasks using iteration. Only one participant chose to solve one task of the interview task (the power task) using recursion. All other participants solved all three tasks of the interview using iteration.

Even though recursion was not the approach the participants used in order to solve the interview tasks, findings from part three of the interview show that participants can successfully solve these tasks using recursion, even through recursion is not their initially selected approach.

Most participants did not write a recursive solution to the tasks presented during the interview. However, when presented with two correct solutions (iterative and recursive) participants were likely to choose the recursive solution over the iterative solution. This was apparent in part two of the interview, where more then 50% of the participants chose the recursive solution to both the factorial task and the binary tree task. As an answer to the research question, participants do not choose to use recursion when
ask to solve algorithmic programming tasks, but they are more apt to choose the recursive solution when given the choice between an iterative and recursive solution.

6.2.1 Common recursive examples: Factorial and Binary Tree. Several researchers suggest that factorial is a very common recursive example that is used while teaching recursion (Ginat & Shifroni, 1999; Morrone, 1995; Wiedenbeck, 1988). It might be the case that a recursive factorial example was used in class while teaching the topic of recursion. This may have contributed to the reasons why participants chose the recursive solution for the factorial task. Factorial is considered a classic recursive example, used by most educators to teach the topic of recursion (Henderson & Romero, 1989; Kruse, 1982; Wirth, 2008).

A binary tree is often used as an example of a recursive structure and as an example for writing recursive functions. Michail (1996) proposed a new way to teach binary tree algorithms through visual programming. The findings suggest that visual programming is an ideal way to teach data structure algorithms such as a binary tree. Counting the nodes of a binary tree could be used in class while teaching recursion. The researcher speculates that the reason participants in the present study chose the recursive solution to the binary tree task is because this solution was used or shown in class while teaching the concept of recursion. In addition, participants might associate the binary tree structure with a recursive structure and therefore might be more likely to choose a recursive solution for a recursive structure. Data from part four of the interview is used to examine the relationships between defining the structure of the task to the approach used or chosen to solve it. These results are presented and discussed later in section 6.4.
6.2.2 Reasons for choosing iteration: Class related. As mentioned previously, findings show that participants do not choose to use recursion when asked to solve algorithmic programming tasks. After solving the first three tasks (the power task, the summation of digits task, and the search list task) using both approaches, the researcher asked each participant to discuss the main reason why he or she initially solved the task using their chosen approach. Chapter 4 presented the complete listing of responses categories with the frequency with which each category appeared. Four categories were identified: “stack overflow,” “properties of recursion,” “class or teaching related reason,” and other. Two out of the four categories presented, “stack overflow” and the “properties of recursion,” were discussed previously. Eight participants’ responses were categorized as “class or teaching related reason.”

Participants’ responses that were categorized in this category contained a discussion of the order of the concepts taught within the computer science curriculum. In particular, the reason why participants initially solved the tasks using iteration may be because when learning to program, iterative constructs such as for loops and while loops were taught first, and only later during the course the concept of recursion was taught. Participants believe that because iteration was taught first, this is the approach they choose to use. Research has shown that that teaching recursion before iteration leads to deeper learning of programming and students tend to leave the first programming course with better problems solving skills than if they are taught iteration first (Mirolo, 2010; Turbak & Royden, 199AD; Wiedenbeck, 1988).

Participants in current study were taught iteration prior to recursion, which might lead them to believe that iteration is the approach they are more apt to turn to when
solving programming tasks. It can be inferred that students believe that because they learn iteration prior to recursion, they are more familiar with iteration rather than recursion. Therefore, they are not turning to recursion due to lack of practice using recursion. However, findings from this study show that most of the participants were successful in solving programming problems using recursion, even though they learned iteration prior to recursion.

6.2.1 Recursive solution and base case. Part one of the interview asked participants to solve the power task, summation of digits task and the search list task without prompting about which approach to use. Part three asked the participants to solve the same task using the other approach. For example, if during part one a participant solved the power task using iteration, during part three the participant was asked to solve the task using recursion. Findings show that during the first part of the interview participants solved the three tasks using iteration, even though findings from part three show that participants can successfully solve these tasks using recursion. To answer the research question, participants do not choose to use recursion when asked to solve programming tasks.

It is worth noting that in order to halt the execution of the recursion function, one must create a base case. Research shows that students have various difficulties with the concept of recursion (Haberman & Aviv, 2002; Lahtinen, Ala-Mutka, & Järvinen, 2005; Velázquez-Iturbide, 2000). When writing a recursive function, one must construct and write a base case in order to halt the execution of the function. Research has shown that students have difficulties with recursion, in particular regarding the base cases of
recursive functions (Haberman & Averbuch, 2002). Haberman and Averbuch claim that the difficulty with the base case might affect how students perceive the correctness, readability, and code complexity of recursive functions. Even though students must split the problem into subproblems and identify the relevant base case(s), Haberman and Averbuch claim that students have difficulties in making that distinction and usually use redundant base cases or ignore the need for an essential stopping conditions. Due to the fact that most of the participants chose to solve the tasks using an iterative approach, part three of the interview required the participants to solve the same tasks using recursion. Chapter four contains a discussion about the various base cases of each task. Most participants did not present difficulty identifying the base case of the task in current research

When solving the power task recursively, one participant ignored the need for a base case in order to halt the function. In particular, he or she created an infinite recursive function, which was categorized as an incorrect response to the power task.

When solving the summation of digits task recursively, none of the participants made an error in the base case part of the response. Three participants did not attempt the task, and it could be that because they did not identify the base case, they choose to not attempt the task. When solving the search list task recursively, six participants did not attempt to solve the task. As discussed in Chapter 4, search list task required two different base cases, which means this task might be more difficult than the power task and the summation of digits task. When calculating $x^n$ using recursion, one is calculating $x \cdot x^{n-1}$. In order to halt the recursion, one can stop the recursion when the exponent is equal to zero, calculating the case of $x^0$, which will return 1. Another correct base case
would be when the exponent is equal to one, calculating the case of $x^1$, which will return $x$. The base case $x^0$ will require one more recursive call over the base case of $x^1$. When returning the base case, the result will be multiplied by the previous function call, therefore there is no benefit or drawbacks regarding the result of the function when reaching the base case of $x^0$, which will multiply the previous function call by 1.

However, when solving the power task using recursion, one will need to use either the base case $x^0$ or the base case $x^1$, but not both. Using both cases will create a redundant stopping condition. If the function will stop at $x^1$ and return $x$, the recursion will never get to the next step, which is when $x^0$, therefore including both base cases, in the same function, is not necessary. However, when solving the Search List Task using recursion one must use two different base cases. The list task also asks participants to write a function that will yield “true” if the item is in the list and otherwise yield “false.” The recursion can be stopped using two different scenarios. The first scenario is when the base case compares the given item with the current item of the list: if they are equal then the function should return true and halt. If the item is not found, meaning the function compares every item from the list and it was not a match with the given item and now we are at the end of the list, or the list is empty, the function should return false and halt the recursive function. In order to halt the function, the second base case should return false when there are no more items in the list or if the list is empty. Both base cases need to be used together in order to successfully solve the list task. When a task requires a combination of two or more different base cases it might be considered of a higher difficulty then a single base case task. Thus, one might claim that the list task was of a higher difficulty than the power task.
Due to the difference in difficulties, it can be speculated that more participants did not attempt the list task because they failed to identify the two different base cases for the search list task. In summary, even though participants did not appear to have major difficulties with constructing the base case in the power task, it may be that the identification of two different base cases to solve the list task prevented the participants from writing an attempt response to the list task. However, there is no evidence to support this claim.

6.3 Participants modeled the tasks using different models: Results of part four of the interview

Part four of the interview occurred after participants were asked to solve the interview tasks using the approach of their choice (iteration) and prompted to use recursion. During the fourth part of the interview, participants were asked to discuss how they modeled each of the tasks. For the purposes of this study, “modeled” means that the participants were asked to discuss how they thought of the task prior to solving it, or how they defined the task for themselves. Each task in the interview could be modeled using an iterative model or a recursive model. For example, \( x^n \) can be modeled iteratively as \( x \cdot x \cdot x \cdot ... \cdot x \), \( n \) number of times, or it can be modeled recursively as \( x \cdot x^{(n-1)} \). Another example is factorial, which can be defined iteratively, and recursively. Five factorial can be computed iteratively as \( 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \), or it can be defined recursively as \( 5 \cdot 4! \)!

Examples of the recursive model and the iterative model for each task were presented in chapter three.
The participants were shown two different postcards with the two different models of the question. The goal for this part was to examine if there is any relationship between the way the participants modeled or defined the task and their initial solution to the task. Only 13 participants were asked to respond to this part of the interview. Even though the population of this study included 17 students, this part of the interview was added after 4 participants already completed the interview tasks. Therefore, the sample set of this part was only 13 participants total.

Results suggest a relationship between modeling the task and the approach taken to solve the task. Since the modeling follow-up questions were asked after participants already solved or attempted to solve the first three tasks using both approaches, and were presented with both the iterative and recursive solution to the other two tasks, there might be a bias with the results of this part of the study.

When asked if they modeled the power task as an iterative or recursive task, 12 participants responded that they thought of the power task as an iterative task. Meaning, when calculating $x^n$, the result should yield $x \cdot x \cdot x \cdot x$, $n$ number of times. Only one participant modeled the power task as a recursive task. It is worth noting that the participant who solved the power task using recursion during the first part of the interview without prompting, modeled the power task as an iterative formula.

Results of the digit task mimic the results of the power task, where 12 participants responded that they thought of the digit task as an iterative task and one participant modeled the task as a recursive task. When participants were asked how they modeled the list task, all 13 participants responded that they modeled the task as an iterative task. Hence, list items are individual items, and it is not the first item of the list connected to
the rest of the list. Figure 6.2 shows that a list can be considered to be a recursive structure. The list consists of multiple sub-lists. In Figure 6.2, item 1 is the first item of the list and items 2 and 3 represent the sub-list.

![Figure 6.2 Example of a list as a recursive structure](image)

Even though over 90% of the participants modeled the first three tasks iteratively, results were different for the two tasks in part two of the interview. Part two of the interview contained two tasks, the factorial task and the binary tree task. When asked about the factorial task, ten participants responded that they modeled the task as an iterative task, whereas three participants modeled the task recursively. This means that 76% of the participants modeled the factorial task as $n \cdot (n-1) \cdot (n-2) \ldots \cdot 1$. On the other hand, nine participants modeled the binary tree task as a recursive task, where four participants modeled the binary tree task as an iterative task.

The interviewer only asked participants to state how they modeled the task prior to solving it without any follow-up question, therefore there is little evidence to show a connection between the participants’ solutions and their responses to how they modeled the tasks.
6.4 Discussion of Participants modeled the tasks

There is a difference between the recursive call within the function and the recursive object of the task. Thompson (Thompson, 1985) hypothesize that in order to be able to recognize a problem as a recursive problem, one should recognize the object of the tasks as have a recursive structure. Meaning, in order for a student to write a recursive solution to the factorial task, he or she should define factorial as a recursive definition:

\[ n! = n \cdot (n - 1)! \]

This research is consistent with the results of the current study. During the first part of the interview, participants solve the tasks using iteration, and during part four of the interview, participants identify the objects of the tasks as iterative objects.

The conclusions, implications for teaching and further research will be discussed and presented in the next chapter.
CHAPTER 7 – CONCLUSIONS, IMPLICATIONS & SUGGESTIONS FOR FURTHER RESEARCH

This chapter contains conclusions from this research, implications of the findings, and suggestions for further research. First is a discussion of what the findings of the research say about students’ use of recursion when solving programming tasks. Second, implications for instruction are discussed, and lastly suggestions for further research are offered.

7.1 Students do not chose to solve algorithmic programming tasks using recursion.

Findings from the present study suggest that the answer to the question of whether students choose to use recursion is no. During the interviews, when asked to solve the first three tasks without prompting about which approach to use, participants chose to use iteration. Only one out of 17 participants chose recursion as a solution approach to any of the three programming tasks.

Ginat (2004) also concluded that the majority of his study’s participants do not approach programming tasks recursively. Instead, they attempted to solve programming tasks using various other approaches, which led to errors and inefficient solutions. Even though participants in Ginat’s (2004) research had learned and seen a variety of recursive solutions while learning about the concept of recursion, recursion was not their chosen approach. Although the current study and Ginat’s (2004) study used different tasks to gather data, the same conclusions can be drawn. Ginat’s research showed that students do not approach programming tasks using recursion when recursion might be a suitable
approach to use, Giant did not investigate if participants who did not approach the task using recursion, *could have* successfully written a recursive function. As an addition to the findings from Ginat’s (2004) research, most participants in the current study were able to solve all three programming tasks correctly using recursion, even though it was not their initial chosen approach. It can be speculated that the difference between Ginat’s study and current study might be due to the different research design and data analysis methods. The current study involved interviewing 17 participants, during which they were asked to solve programming tasks. In Ginat’s study, students solved the tasks and then some participants were interviewed about their solutions at a later point.

Unfortunately, Ginat did not present the data analysis methods within his research article, therefore it is difficult to know what the specific differences are between this aspect of these studies. It could be that the data analysis methods differences contributed to the differences in the findings, but because of the lack of available information, it is not possible to compare this aspect of the two studies. In addition to the possible difference in data analysis methods, Ginat used optimization and counting tasks, which are different type of tasks from the tasks in this present study. It is possible that for the participants in Ginat’s study, those tasks were more difficult or less familiar than the tasks used in the current study were for those participants.

7.2 Students can write recursive solutions when prompted.

Even though research suggests that recursion is a difficult concept (Hsin, 2008; Levy, 2001; Morrone, 1995; Pears, Seidman, Malmi, & Mannila, 2007; Turbak & Royden, 199AD; Wirth, 2008), findings from current study show that most of the
participants can successfully solve these programming tasks using recursion; however, they do not use recursion as solving tool without prompting.

Researchers have proposed that students have difficulties identifying the base case of recursive functions (Haberman & Aviv, 2002). Findings from this study, however, show that participants are successful at solving tasks using recursion and identifying the correct base case. When multiple base cases were required for a particular task, however, significantly more participants did not attempt to write a recursive solution. This suggests that for participants in this research, tasks with multiple base cases might be more difficult and they might be less inclined to use recursion to solve those kinds of tasks.

Although the answer to the research question about whether students use recursion was no, when prompted to approach the problem using recursion, they can successfully do so. The rest of the study’s research questions were focused on understanding why students choose not to use recursion even though they were capable of successfully writing recursive solutions to the interview tasks.

7.2.1 Reasons students solved tasks using iteration. One of the goals of this study was to reveal why participants did not initially solve the task using recursion and why iteration was their choice of approach. Participants discussed various reasons for not choosing to use recursion as their initial approach. These reasons fell into three general categories: Beliefs about recursion, beliefs about how they were taught and beliefs about the challenges of writing a recursive solution. Some participants hold a belief that recursion will often cause stack overflow, even though that is not the case. Stack overflow can be caused by infinite recursion however, participants in this study tended to
over generalize and avoid recursion, even though recursion might be a viable and appropriate solution.

All participants in this study were taught iteration prior to introducing recursion. Some participants believe that they are drawn to iteration because of the sequence of concepts taught to them. In other words, they were taught iteration first so they are choosing to approach the tasks using iteration, perhaps because of their familiarity with it.

Other participants said they would prefer to use iteration because writing a recursive solution would be more difficult than writing an iterative one. Although many students showed a preference for recursively written solutions, they expressed that actually generating a recursive solution would be more difficult for them.

As discussed in previous chapters, stack overflow often occurs in the case of infinite recursion. When using recursion, during every recursive call, a new stack call is created. When a new stack call cannot be created, a stack overflow error occurs. Results show that only one participant wrote an infinite recursive solution (which is the type of solution that creates stack overflow) as a solution to one of the programming tasks and did not attempt the other two recursive tasks. Participants believed that they chose to solve programming tasks using iteration to avoid stack overflow. However, the researcher tested various inputs using the same programming tasks and they did not result in stack overflow. Therefore, it can be concluded that stack overflow is not a valid reason to avoid recursion. There is a lack of research regarding students’ beliefs about stack overflow error but it appears, from the findings, that student hold beliefs about recursion and stack
overflow that discourage them from using this approach even in situations where the belief is not warranted.

Even though recursive solutions are often short and elegant, they are abstract and non-trivial to construct. Reasons for not using recursion that participants gave regarding the length of the solution, readability and abstraction were categorized as “property of recursion.” Some participants claimed that they did not choose to solve the tasks using recursion because the recursive solution was abstract. Research supports the claim that recursion is an abstract topic that is hard to learn as a student (Bhuiyan, Greer, & McCalla, 1994; Kessler & Anderson, 1986; Wiedenbeck, 1988) and that appears to be one feature of recursion that prevents students from using it as a solution strategy.

In addition to discussing the properties of recursion, which include the length, simplicity and elegance of recursion, some participants stated that recursion is hard to visualize. As mentioned in chapter two, the ideal knower should include three different elements in order to construct a successful recursive function. The three different elements are: active flow, which is the recursive call, base case, which is the terminal condition of the function and passive flow, which is the result of the function. The result of the recursive function will be built up from each recursive call. The active flow and the base case are present within the recursive function and must be written but the passive flow is the abstract element within the recursive solution. Some participants stated that when constructing a recursive solution, it was hard to visualize the recursive solution. For example, one participant said:

it takes a while for me… It’s a lot harder to visualize I guess because you are going down through the recursion tree, or what not and it’s really hard to like…
when you have numbers like 10 if you decrementing by 1 to start at those are then recurse through them in your head, but with a loop it’s a lot easier for me to see I guess.

This participant mentioned that it is harder to visualize the progression of the numbers in recursion because each recursive call is being execute in a certain order and it is hard to follow where you are in the recursion without a tracing chart. However, participants felt that when using iteration, it is a lot easier because you can tell where you are in the loop.

To summarize, students’ verbal responses can be considered to be of two type of reasoning. One type of reasoning is incorrect beliefs about recursion and over generalization of this instance (stack overflow). The other type of reasoning is based upon (correct) impressions students have about recursion. This suggests that teachers who want to improve learning of recursion could address the problem in one of two ways: one is to work to counter the incorrect beliefs and present instances in the classroom where these beliefs do not hold true. The other suggestion is to help students understand recursion better, especially the elements of recursion that can be so difficult, and present the students with various example to enhance their confidence.

Even though researchers have concluded that recursion is a very difficult concept to teach (Bhuiyan et al., 1994; Haberman & Aviv, 2002; Henderson & Romero, 1989; Lahtinen, Ala-Mutka, & Järvinen, 2005) some researchers suggest that in order help students understand recursion better one can teach recursion by presenting different visual examples such as fractals images, visual path of finding a solution to a maze, or using the Russian Nesting dolls. (Dann, Cooper, & Pausch, 2001; Stephenson, 2009; Stern & Naish, 2002). Research suggests that using graphical representation can help
introduce fundamental concepts such as recursion and help with the level of abstraction of this topic. Research also suggests that using three-dimensional animation programming environments, such as Alice, to teach recursion can be beneficial due to a combination of visualization, experimentation and mathematical explanation of the concept. Alice is an innovative programming environment that allows students to learn fundamental programming concepts in the context of creating animated movies and simple video games (Dann et al., 2001).

Hsin (2008) proposed to help students understand the process of recursion by using RGraph or recursive graph. Hsin used Towers of Hanoi as an example to show how to construct an RGraph layer by layer. Hsin claimed that a visual aid, such as an RGraph, can help students understand the flow of a recursion process. The first and the second stages of the Tower and Hanoi task using recursive graph are presented in Figure 7.1.

![Figure 7.1. RGraph example using stage one and two of Tower of Hanoi task](image)

The graph can help a student visualize when a certain recursive call is being executed and in what order, and what is the return value of it. Using visual aids can help a student see the passive flow of the recursive function, and help make the concept of
recursion less abstract. In addition to Hain (2008), Stern and Naish (2002) found that visual representations are suitable for particular algorithms and can help students construct a viable mental model of the process of recursion.

Some participants believed that because the curriculum introduced iteration prior to recursion, they are choosing to approach the tasks using an iterative approach. This reasoning was categorized as “curriculum related or teaching related.” Results from Wiedenbeck (1988) showed that even when recursion is taught before iteration, students tend use iteration as a tool to solve programming tasks. The current study found that students who were taught iteration first choose to implement programming tasks using iteration, and they believe that the order of teaching this concept is the reason of implementing programming tasks iteratively. All participants in this study were taught iteration prior to recursion. These findings support Wiedenbeck (1988) result that students do prefer to use and implement iterative solution to programming tasks. Additional research would be needed to determine whether there is a significant impact on students’ preferences if the (less standard) order of instruction was used (with recursion taught first).

To summarize, findings from the current study suggest that students do not use iteration as a tool to solve programming tasks and furthermore, they believe that a recursion solution will always yield stack overflow, even though that is not the case. Even though stack overflow could occur when writing an infinite recursive solution, students tend to over generalize the occurrences of stack overflow, and avoid recursion due to this belief. This is an incorrect belief about recursion that should be address while teaching the topic of recursion. Another reason participants stated was that the
curriculum prompted them to use iteration due to the sequence of topics. Students believe that the sequence of teaching concept drawn them to use iteration instead of recursion, because iteration was taught first.

7.3 Students prefer the recursive solution when given a choice between two different solutions

While the aim of this study was to investigate when students use recursion and why, the researcher wanted to investigate if results would differ if students were asked to construct a solution verses asked to pick a solution. Two of the interview tasks presented a correct iterative solution and a correct recursive solution and participants were asked to pick their solution of choice. Findings show that students do not turn to recursion when asked to construct a solution to a programming task, but when asked to pick between an iterative solution and a recursion solution, it they preferred the recursive solution. This shows that participants of this study might not think of recursion when solving programming tasks, but, if presented with two solutions, one iterative and one recursive, they will prefer the recursive solution to the iterative solution.

The two interview tasks for this part were to calculate factorial and to count the nodes in a binary tree. Research have shown that both factorial and binary tree are examples that are usually used when first introducing recursion (Kurtz & Johnson, 1985; Michail, n.d.; Morrone, 1995; Turbak & Royden, 199AD). It might be the case that these tasks prompted the participants to choose the recursive solution because these tasks were used or discussed while teaching recursion. Participants preferred the recursive solutions,
and they discussed the simplicity and elegance of recursion. This study shows that students find the length of the recursive solution appealing.

Even though there are no strong evidence, this study suggests that students might be able to recognize the feature of the recursion solution and prefer it as a solution for the recursive example. This suggests that by solving and implementing various examples when teaching the topic of recursion, students are developing some understanding of the benefits and useful features of using recursion. Students in the present study seemed to understand the value of recursion. The challenge is to get them to value it enough that they will choose to use it. Teachers should try to emphasize the beneficial features of the recursive solution, such as the length, simplicity and elegance. It might be the case that if teachers emphasize the benefits of the recursive solution more, it would prompt students to implement and write recursion solutions. A future research study could investigate whether students who were taught by using modified instruction that is focused on the benefits of recursion with various recursive examples employ recursion more frequently when solving programming tasks.

Research shows that calculating factorial and counting the nodes in a binary tree are common examples used when introducing recursion (Bruce, Danyluk, & Murtagh, 2005; Ginat & Shifroni, 1999; Haberman & Aviv, 2002; Henderson & Romero, 1989; Kruse, 1982; Leron & Zazkis, 1986; Morrone, 1995; Turbak & Royden, 199AD). The factorial task and the binary tree task were used during the interview when participants had to choose between an iterative and a recursive solution. Therefore, it is possible that the factorial task and the binary tree task prompted participants to choose recursive solutions because of their association of these particular tasks with recursion.
Researchers have shown that students do not turn to recursion to solve programming tasks (Ginat, 2004) however, there is lack of research regarding students solution of preference. Findings from the present research add to previous research by concluding that students choose recursive solutions when presented with them.

### 7.3.1 Reasons students chose a recursive solution.

Results show that when presented with an iterative solution and a recursive solution, students prefer the recursive solution. A secondary goal of the current study was to reveal why participants prefer recursive solutions to iterative solutions. Participants discussed various reasons for choosing the recursive solution to the factorial task and the binary tree task during the interview. Participants’ verbal responses were analyzed and four main response categories emerged: “stack overflow,” “properties of recursion,” “lack of confidence in solution” and “structure resemblance.”

Even though most participants preferred recursion as a solution to the factorial task and the binary tree task, some participants chose the iterative solution. Participants who did not choose recursion reasoned about it by stating that they would like to avoid recursion in order to avoid stack overflow. As mentioned previously, using recursion can cause stack overflow for large input values, however when the interview tasks were tested the recursive solution did not cause stack overflow.

The predominant category of reasons was “properties of recursion.” Participants were more apt to choose the recursive solution because it is short, elegant, simple and appealing. As mentioned previously, every recursive task can be implemented using iteration, however, sometimes a recursive solution can be shorter, more elegant and
simpler than the iterative solution. One example is the Binary Tree Task. During part two of the interview, participants were presented with an iterative and a recursive solution. The iterative solution was longer in length and more complicated because used a stack structure to hold the data from the nodes of the tree. Simplicity, elegance, and readability are all properties of a recursive solution. Participants preferred the recursive solution to the iterative solution because of its properties even though both solutions could be used to solve the task.

For certain tasks, a few participants were not able to follow a solution, either the iterative one or the recursive one. Due to this, they chose the solution that they were able to follow and understand. In other words, if participants were not confident that the iterative solution solved the task successfully (even though they were told that all the solutions were correct), he or she would choose the recursive solution or vice versa.

Some participants reasoned that a binary tree is a recursive structure and therefore they chose the recursive solution to solve the binary tree task. As discussed in Chapter 6, binary tree can be defined as a recursive structure because a binary tree consists of multiple sub-trees that each has a similar structure to the binary tree.

As mentioned previously, there is little to no research regarding students’ solution preference, therefore, when given two solutions and asked “which solution would you choose and why?” this study revealed various reasons why students chose a recursive solution over an iterative one. This can be considered as an addition to the computer science education research field. Knowing why students prefer a recursive solution to an iterative one can help us understand how students think about the features of recursive solutions. This current study revealed that students prefer the recursive solution because
of its simplicity and elegance. Knowing what belief students have about the features of recursion can help improve our teaching practices. Instructors can emphasize the positive and correct beliefs and try to dispel the incorrect ones. For example, if a recursive solution is simple, elegant and less complex than the corresponding iterative solution, as teachers we should emphasize this positive feature to encourage students to use recursion in those situations. On the other hand, if students believe that recursion often causes stack overflow, as teachers we should try to use examples that dispel this false belief. Teaching tail-recursion might also help to dispel this false belief. Tail-recursion is a recursive function where the result is built while the recursive call is implemented which is different then the passive flow which is building the result after the recursive call have reach the base case. Inherently, tail-recursion will not cause stack overflow because the recursive calls do not take up additional stack space. Even though finite recursion will not cause stack overflow, this current study showed that students over generalize and believe that recursion will cause stack overflow. These approaches may help build their confidence in using recursion without causing this to happen.

In conclusion, participants in the current study preferred recursive solutions to iterative solutions when asked to choose a solution to calculate factorial or to count the nodes in a binary tree. As noted previously, both factorial and binary tree tasks are common recursive examples used in instruction; it is possible that participants are more apt to choose a recursive solution when the task is a commonly seen recursive task. This will provide us insights about students understanding recursion, how they evaluate a recursive solution and what features of the recursive solution they drawn to.
7.4 Participants modeled the tasks using different models

Even though the main goal of this research is to investigate when student use recursion as a tool to solve programming tasks and why do they choose to do so, the results show that it is also vital to investigate how to students think of the tasks prior to solving them. The reasons participants gave for choosing recursion as a solution prompted the researcher to investigate how students initially thought of each task that was asked during the interview.

Findings show that each task yielded different results. The majority of the participants in this research defined the first four tasks (Power task, Summation of Digit task, Search List task and Factorial task) as iterative tasks. However, over 70% of participants defined the Binary Tree Task as a recursive task.

Thompson (1985) claims that in order to be able to write recursive procedures or functions, one must first learn how to describe the object as a recursive structure. The current study shows that most participants define the Binary Tree Task as a recursive task, and they also chose the recursive solution as their preferred solution to this task. Even though the evidence is not very strong, it can be speculated that students chose the recursive solution because they defined the task recursively. Even though Thompson (1985) does not discuss participants choosing a solution, it might be the case that if students define the task using a certain method, they would also chose that approach as a preferred solution. In addition, the current study shows that participants can successfully solve programming tasks using recursion even though they might not initially define the task as a recursive task.
7.5 Limitation of study and further research

An interesting aspect of this study is that one of its strengths also happens to be its primary weaknesses: the interview instrument. This study was conducted in order to study when and why student use or do not use recursion when solving programming tasks. Once findings showed that students do not spontaneously use recursive, the researcher investigated whether students could write a recursive solution successfully. Prior to prompting students to use recursion, an example of a recursive solution was presented to them. Even though participants were not allowed to refer back to the example when writing their own recursive solution, the fact that they saw a recursive example could be considered to be a limitation of this study. It might be the case that participants gain knowledge about the construction of the recursive solution by looking at and examining one during part two of the interview. Therefore, it can be speculated that because students saw a recursive solution it may have helped them to construct a recursive solution in the next part of the interview.

Secondly, conclusions drawn concerning the use of recursion were based upon a small sample size of 17 participants. Although some interview tasks were based upon previous research studies, the interview instrument given to the participants had not been tested for validity and reliability before being put into practice, and this might be considered a limitation of this study. Due to the small sample size we cannot generalize these results and claim that is it the case that all students do not choose to employ recursion when asked to solve programming tasks. In addition, there might be additional reasons for initially approaching the task using iteration. A larger sample size could reveal additional reasons that did not appear while conducting this study. In order to
achieve strong conclusions about students’ use of recursion, studies with larger sample sizes should be conducted using current interview instrument.

Even though this study answered the research questions about students understanding and use of recursion, many questions arose that require further research. All participants of this study learned iteration prior to recursion; it would be worth investigating whether a different sequence of teaching would yield different results. Knowing what is different in students’ understanding of recursion when teaching programming course using different concepts sequences (such as iteration prior to recursion or recursion prior to iteration) might help the field of computer science education to enhance the curriculum, and help students gain a deeper understanding of recursion. In other words, if we know that teaching recursion prior to iteration can motivate students to use recursion appropriately and successfully, that information can benefit the field and help design an improved curriculum.

In addition, we might want to investigate how students think of the interview tasks prior to solving them. As a limitation of this study, participants were asked how they define the task after solving the tasks using both iteration and recursion. It might yield different results if this aspect of the interview were presented to participants prior to solving programming tasks. As Thompson (1985) suggests, students might be able to solve and implement recursion solutions if they consider the tasks to be recursive tasks. If we find out how students define the tasks prior to solving them, one can make a connection between the approach they implemented and the approach they used when they defined the task. If the approach implemented and approach defined are related to one another, it can be an additional support to Thompson’s claim. However, if there is no
connection between the approach implemented and the approach defined, it can weaken Thompson claim and further research might be needed to order to reveal why this is the case.

Another research study could be conducted twice: once after students finish their first programming course, and once as they finish their degree. It would be interesting to investigate if their approaches to solving such programming tasks changed while learning about the different recursive algorithms and recursive structures. It would help the field understand the amount of knowledge gain that students develop throughout the computer science degree curriculum. It would be interesting to investigate what skills students gain between starting and finishing their computer science degree, particularly regarding solving programming tasks using recursion.

In conclusion, this study found that students do not choose to use and implement recursion when solving programming tasks. However, they do prefer the recursive solution to the iterative one. These finding can inform and benefit the teaching practices of educators when teaching recursion. In addition, future research in this area, revealing more about students’ understanding of recursion, can help the Computer Science education community understand what will help students better learn the important topic of recursion. This can improve teaching practices, retention rates in college degree program and provide better computer programmers to society to help improve technology related fields.
REFERENCES


BIOGRAPHY OF THE AUTHOR

Adi Levy Conlogue was born in 1985 in Tel Aviv, Israel. She graduated from Kalay High School at Givatayim in 2003 and served a year and a half in the Isreali Defense Force. She moved from Israel to Maine, USA in 2005 in order to start her Bachelor degree and swam as a division I student-athlete at The University of Maine. She received her Bachelor’s Degree in Computer Science from The University of Maine in 2009. She attended graduate school at the University of Maine from 2010 to 2013, where she studied Mathematics and Computer Science Education Research as part of the Masters of Science in Teaching program. In the summer, Adi usually flies to Israel for a month to visit her family. This summer Adi is planning to move to sunny California to work as a high school teacher. Adi enjoy spending time with her friends and family, and playing video games with her husband, Ryan. Adi is a candidate for the Master of Science in Teaching degree from The University of Maine in May 2013.