Bachelor's Thesis (UAS)

Information Technology

Telecommunication and Networking

2012

ANH DUONG (DƯƠNG ĐỨC ANH)

COMPUTER SCIENCE

RECURSION

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES
Information Technology | Telecommunication and Networking
20/01/2012 | 50 pages
Instructor: Hazem Al-Bermanei

DUONG DUC ANH

COMPUTER SCIENCE - RECURSION

The purpose of this thesis is to analyze recursion in the field of computer science field. There are three sections in this thesis which are introduction, examples of using recursion and applying recursion.

The first sectionintroduces recursion and related topics from general to details. First of all, definitions of computer science, algorithm, mathematical induction and recursion are introduced. Then, recursive function, algorithm and data types are discussed. Finally, there is the classical comparison between iterative and recursive methods.

The second section contains three examples of using the naïve recursive method and recursion's developed branches such as the divide and conquer algorithm and dynamic programming. Those examples are the eight queens puzzle and finding the maximum and the longest increasing subsequence.

The third section demonstrates the full potential of using recursion with functional programming to make the code more concise and easier to understand. In this section, the examples of the quicksort algorithm are shown in both types (functional and imperative) of programming.

In this thesis, all algorithms are described by using functions written in a certain programming language. The algorithms in the first and the second sections are described by functions written in Java. The choice of Java programming language here is based on the fact that Java is very popular and easy to understand. In the last section, the functions are written in the Scala programming language. The Scala language was chosen instead of the Java language because Scala supports both the imperative style and the functional style of programming. That feature assists in making the comparison more noticeable. In addition to the codes in the examples in the second section, there are also the full source code written in Java and some screenshots in the Appendix which were captured during the process of writing this thesis as proofs to the reality of the functions.

KEYWORDS:

Recursion, dynamic programming, divide and conquer, recursive algorithms, computer science.

FOREWORD

This Bachelor's thesis was written from summer 2011 to spring 2012 at Turku University of Applied Sciences. To complete this thesis, I received supportfrom my family, my teachers, and my friends. I would like to give my thanks to all of them.

Firstly, I would like to thank my mother, father, little brother and my fiancée who supported me a lot on financial issues and motivated me to study well in Finland. I would not be anywhere near this point of completing my thesis without them.

Next, I would like to thank all my instructors in Turku University of Applied Sciences. I would especially like to thank my supervisor Mr. Hazem Al-Bermanei and my software testing course's instructor, Ms. Tiina Ferm for their guidance of writing thesis, to my computer instructor Mr. Tero Virtanen with whom I can share my ideas freely as a friend, and to Mrs. Marjo Joshi and Mr. Balsam Almurrani who taught me the required writing and programming skills respectively to complete this thesis. I also would like to thank my degree program manager, Mr. Patric Granholm, for his flexibility and support he has given to me.

Then, I would like to say “thank you” to my friend, Nguyen Dang Chau, who examines my codes and helps me to improve them. Thanks to him my codes now are a lot better than when I started writing this thesis.

Finally, thanks to anyone who is interested in reading my thesis. I hope it will be useful for you.

20/01/2012, Turku, Finland

Duong Duc Anh

CONTENTS

I. INTRODUCTION **1**
1. Definitions 1
1.1. Computer science 1
1.2. Algorithm 1
1.3. Mathematical induction 1
1.4. Recursion 6
1.5. Recursive function 6
1.6. Recursive algorithm 8
1.7. Recursive data types 14
1.8. Comparison between iterative and recursive methods 15
II. EXAMPLE OF USING RECURSION **17**
1. Backtracking algorithm 17
1.1. Introduction 17
1.2. The Eight Queens Puzzle 18
2. Divide and Conquer algorithm 19
2.1. Introduction 19
2.2. Find the maximum 20
3. Dynamic programming 21
3.1 Introduction 21
3.2. The longest increasing subsequence 22
III. PRACTICAL APPLICATION **23**
1. Introduction of the Scala programming language 23
2. Quicksort algorithm 25
2.1 Introduction 25
2.2 Quicksort function written in imperative programming style 26
2.3 Quicksort function written in functional programming style 27
IV. CONCLUSION **28**
REFERENCES **29**
V. APPENDIX **31**

FIGURES

FIGURE 1: THE SCREENSHOT OF THREE SAMPLE RESULTS OF THE EIGHT QUEEN PUZZLE PROGRAM. 36
FIGURE 2: THIS IS THE SCREENSHOT OF ONE SAMPLE RESULT OF THE FIND MAXIMUM PROGRAM 40
FIGURE 3: THESE SCREENSHOTS SHOW THE PROCESS OF FINDING THE LONGEST INCREASING SUBSEQUENCE USING THE WRITTEN PROGRAM. 45

I. INTRODUCTION

In everyday life, recursion can be noticed in many fields such as language, art, mathematics and computer sciences. However, the most common application of recursion is in mathematics and computer sciences, in which it is known by the method of using recursive functions. Especially in computer programming, the concept of recursion is a basic and important concept to build many necessary algorithms.

1. Definitions

1.1. Computer science

A formal definition of the word "computer science" in Oxford dictionary, which is claimed to be "the world's most trusted"[1] dictionary, is as follows: "the study of the principles and use of computers."[2] To be clearer, computer science is the subject which studies and builds scientific foundation for computer-related topics such as algorithm, computer programming, or information processing. The study of computer science assists in the maintenance on current available applications and the development of future computer program.[3]

1.2. Algorithm

Algorithm is one of the most fundamental concepts of computer science.[4] According to the Oxford dictionary again, the formal definition of the word "algorithm" is: "a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer"[5]

1.3. Mathematical induction

Mathematical induction is the method that proves that a given statements is true with any natural numbers (positive integers). The task is done by proving that

the initial statement in the infinite sequence of statements is correct. Next, we are going to prove that if any statement in that sequence is correct, the next statement is also correct.[6]

For example, proving the sum of the first n square numbers (1, 4, 9, 16, ..., 225, ..., n) is $\frac{n \text{ x } (n+1) \text{x } (2n+1)}{6}$. The sums of the first n square numbers where n = 1,2,3,4,5 are: (let denoted the sums by using the character s)

n = 1 → S(1) = 1 n = 1 → $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ = 1

n = 2 → S(2) = 1+4 = 5 n = 2 → $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ = 5

n = 3 → S(3) = 1+4+9 = 14 n = 3 → $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ = 14

n = 4 → S(4) = 1+4+9+16 = 30 n = 4 → $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ = 30

n = 5 → S(5) = 1+4+9+16+25 = 55 n = 5 → $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ = 55

The results prove the formula S = $\frac{n \text{ x } (n+1) \text{ x } (2n+1)}{6}$ where n = 1,2,3,4,5. However, the formula cannot be concluded on just five results. Therefore, the implementation of mathematical induction is necessary. To prove the statement S(n) true ∀ k ∈ Z^+, two steps occur:

Initial step: the first condition is true with S ↔ S(1) is true.

Inductive step: when S(k) is true, then S(k+1) is proved to be true ∀ k ∈ Z^+**.**

This proof technique can be expressed as a rule of inference as follows:

[S(1)∧∀k(S(k) → S(k+1))] → ∀nP(n) [7]

Now, we can apply a mathematical induction to solve the example:

Initial step: n = 1 → S(1) = $\frac{1x(1+1)x(2x1+1)}{6}$ = 1 ↔ S(1) is true.

Inductive step: When S(k) is true ↔ n = k → S(k) = $\frac{k\ x\ (k+1)\ x\ (2k+1)}{6}$

Then we need to prove:

S(k+1) is true ↔ S(k + 1) = $\frac{(k+1)\ x\ (k+1+1)\ x\ (2(k+1)+1)}{6}$

= $\frac{(k+1)\ x\ (k+2)\ x\ (2k+3)}{6}$

From S(k) = $\frac{k\ x\ (k+1)\ x\ (2k+1)}{6}$ = 1 + 4 + 9 + 16 + ... + k^2

and S(k+1) = 1 + 4 + 9 + 16 + ... + k^2 + $(k + 1)^2$ we have:

S(k+1) = $\frac{k\ x\ (k+1)\ x\ (2k+1)}{6} + (k+1)^2 = \frac{k\ x\ (k+1)\ x\ (2k+1) + 6(k+1)^2}{6}$

S(k+1) = $\frac{(k^2+k)\ x\ (2k+1) + 6k^2 + 12k + 6}{6}$

S(k+1) = $\frac{2k^3 + k^2 + 2k^2 + k + 6k^2 + 12k + 6}{6}$

S(k+1) = $\frac{2k^3 + 6k^2 + 4k + 3k^2 + 9k + 6}{6}$

S(k+1) = $\frac{\left(2k\ x\ (k^2+3k+2\right) + 3(k^2+3k+2)}{6}$

S(k+1) = $\frac{(2k+3)\ x\ (k^2+3k+2)}{6}$

$$S(k+1) = \frac{(2k+3)x(k+1)x(k+2)}{6} = \frac{(k+1)x(k+1+1)x(2(k+1)+1)}{6} \text{ (proved)}$$

S(k+1) is true. Therefore, we can conclude that S is true ∀ k ∈ Z^+.

“Mathematical induction is an extremely important proof technique that can be used to prove assertions of this type. The method can be extended to prove statements about more general well-founded structures, such as trees; this generalization, known as structural induction, is used in mathematical logic and computer science. Mathematical induction in this extended sense is closely related to recursion.” [8]

Mathematical induction can be easily understood by using the image of a domino sequence. If both the conditions which are “the first domino is knocked over” and “if the nth domino is down, the $(n+1)^{th}$ domino will be down” true, the domino sequence will be down.

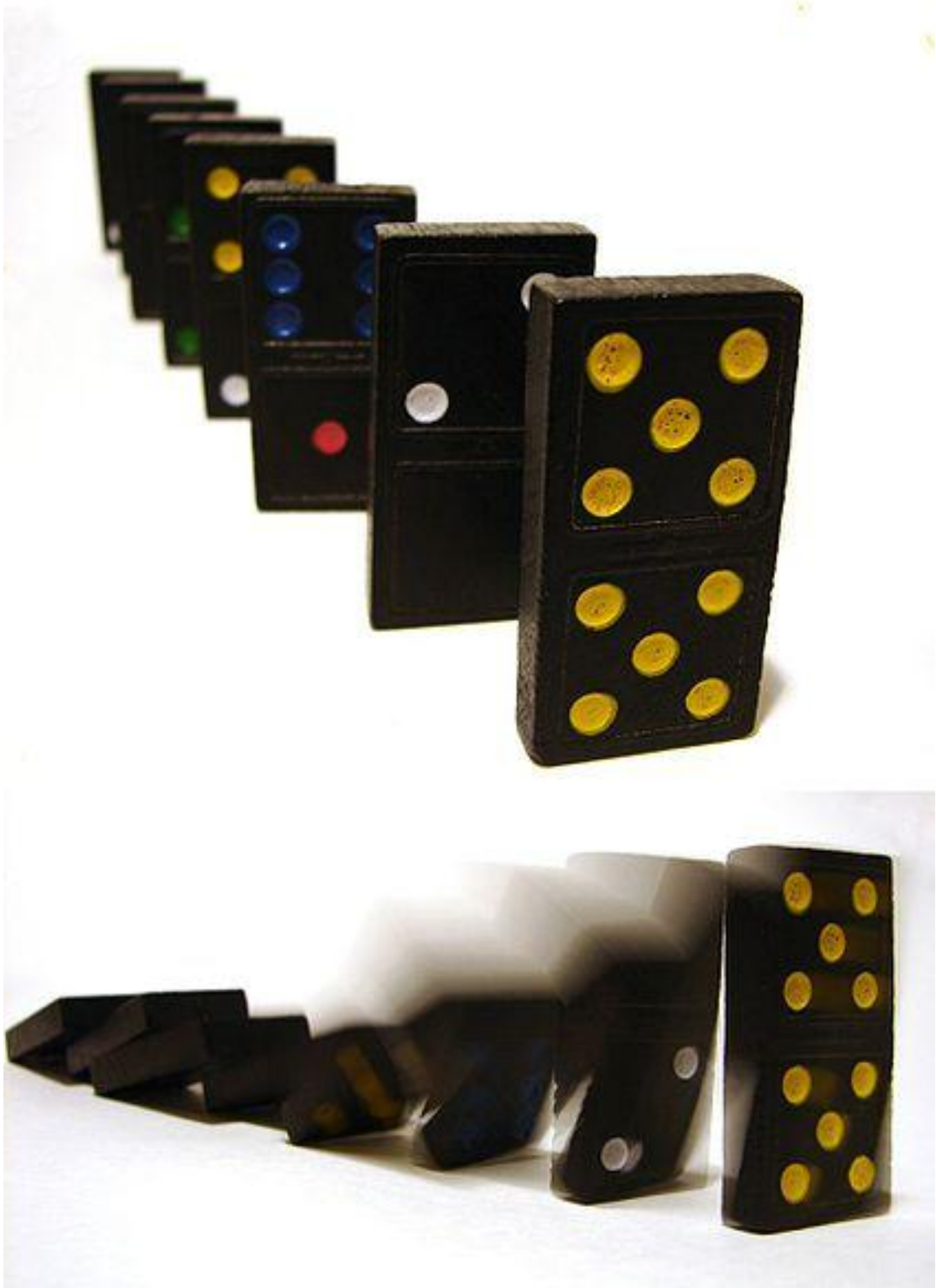


Image 1: Dominoes Sequence [9]

1.4. Recursion

Sometimes it is difficult to give an explicit definition for a certain object. However, that object could be defined easily through the factors of itself. That process of defining objects is called recursion. In simple words, one object is called recursion if it is defined by itself or through another similar object. In mathematics and computer science, recursive algorithms are usually confirmed by using mathematical induction.[10]

Here are some good and easy-to-understand examples of recursion:

Example 1: "When the surfaces of two mirrors are exactly parallel with each other, the nested images that occur are a form of infinite recursion."[11]

Example 2: The sequence of the powers of 3 is given by $a_n = 3^n$ where n = 0,1,2, …. This sequence can be explained by using recursion as follows:

$a^0 = 1$, and the later number in the sequence can be found by using the previous one, $a^{n+1} = 3a^n$.

1.5. Recursive function

A recursive function is a function that can call itself and can be defined by two steps:

Anchor step:

This is a simplest case of the function which can be solved directly and quickly without using any other functions. This step is very important because it prevents the process from infinite repetition.

Recursive step:

The implementation of the function at more complex levels is very difficult or impossible to solve directly. This function will be included in a process of

recursion to recall smaller functions. The process will stop when the original function is reduced to the anchor function. After all solutions of smaller functions are found, we can combine all those solutions to reach the solution of the original function.

Recursive functions are used in recursive algorithms to solve problems which can be solved by recursive solutions. Following are some common examples of recursive function:

Example 1: Definition the recursive function f(n) = n x 2, with n are non-negative integers:

Anchor step: f(0) = 0, when n = 0, n x 2 = 0

Recursive step: f(n) = f(n-1) +2, this is an application of n x 2 = 2 + 2 + 2 + 2 + ... + 2 (the summary contains n numbers).

Example 2: Definition of function f (n) = a^n, with a is a real number different from 0 and n are non-negative integers:

Anchor step: f(0) = 1, when n = 0, $a^n = a^0 = 1$.

Recursive step: f(n) = f(n - 1) x a, $a^n = a^{n-1}$ x a.

Example 3: Definition of factorial f(n) = n!:

Anchor step: f(0)=1, 0! by definition is equal to 0.

Recursive step: f(n) = F(n – 1) x n, n! = (n – 1)! x n.

Example 4: Definition of the summary of n non-negative integers sequence from 1 to n which is f (n) = 1 +2 +3 +4 +5 + ... + n:

Anchor step: f(0) = 0, n = 0 which means the sequence does not contain any numbers. Therefore, the summary must be 0.

Recursive step: f(n) = f(n-1) + n, the summary of n numbers equal to the summary of n-1 numbers plus n.

Example 5: Definition of Fibonacci sequence:

Fibonacci sequence is demonstrated here: 1, 1, 2, 3, 5, 8, ...

Anchor step: f(1)=1, the first number is 1.

f(2) = 1, the second number is 1.

Recursive step: f(n) = f(n-1) + f(n-2), the nth is equal to the sum of the next two previous numbers in the Fibonacci sequence.

1.6. Recursive algorithm

If a problem A is solved by solution of problem A' which is similar to A, it is a recursive solution. The algorithm which corresponds to the recursive solution is called recursive algorithm.

Note: A' must be simpler than A. The solution of A' must be easier to find than A's and not depend on the solution of A.

This is a method that can be used to solve many mathematical problems. Following are examples of the problem to be solved by recursive algorithms. The algorithms are presented by using the Java language.

Example 1: The algorithm to calculate n factorial:

Factorials can be used to count the ways of arranging objects. There are *n*! different ways of arranging *n* distinct objects into a sequence.[12] Factorials also can be used in calculating permutation and computation. In the book entitled “Algorithms in C++”, Robert Sedgewick has described this algorithm as following (in his description he uses N instead of n, but they are the same in this case):

“This recursive function computes the function N!, using the standard recursive definition. It returns the correct value when called with N nonnegative and sufficient small that N! can be represented as an int.”[13]

The following function will get the value of the factorial of the natural numbers n that is passed as parameter to the function.

```
public static int getFactorial(int n){
    if (n==0) return 1;
    else return getFactorial(n-1)*n;
}
```

In this getFactorial function, the anchor step defines the result of the function at n = 0 while the recursive step defines the result of the function at n (n>0) through the value of n and the result of the function at n-1.

For example, the function which is used to calculate 5! can be illustrated as follows:

First, the function defines 5! as follows:

5! = 5 x 4!

After that, the result of 5! is still too complex to calculate directly because of the 4!. Thus, the function defines 4! similarly to 5!:

4! = 4 x 3!

The process continues until it reaches the anchor step and calculates directly the desired result:

3! = 3 x 2!

2! = 2 x 1!

1! = 1 x 0!

0! = 1 (the anchor step is reached)

Finally, the combination of all results has taken place:

5! = 5 x 4!

= 5 x 4 x 3!

= 5x4x3x2!

= 5x4x3x2x1!

= 5x4x3x2x1x0!

= 5x4x3x2x1x1 = 120

Example 2: Euclid's algorithm to find the greatest common divisors of two non - negative integers (one of them must be greater than 0):

"[The Euclidean algorithm] is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."[14]

“One of the oldest-known algorithms, dating back over 2000 years, is this recursive method for finding the greatest common divisors of two integers” - [15]

The following function is created to get the value of the greatest common divisor of the two natural numbers n_1 and n_2 that are passed as parameters to the function.

Note: n_1 and n_2 will be written as n1 and n2 in the following block of codes in order for Java to work.

```
public static int getGcd(int n1, int n2){
    if (n2==0) return n1;
    else return getGcd(n1,(n2%n1));
}
```

This getGcd function starts from the anchor step by determining that when we cannot divide to n_2, n_1 is the divisor. Otherwise, the function will enter the recursive step which goes to find the greatest common divisor of n_1 and the remainder of the calculation n_2/n_1 by using the same function. The found result is the same as the greatest common divisor we need to find from the beginning. That process continues occurring until the anchor step is reached.

For example, the function can be used to calculate the greatest common divisor of (7,5) as follows:

The process starts entering the recursive step and when the anchor step is reached, the greatest common divisor will be returned.

getGcd(7,5)

= getGcd(5,(7%5)) = getGcd(5,2)

= getGcd(2,(5%2)) = getGcd(2,1)

= getGcd(1,(2%1)) = getGcd(1,0) (the anchor step is reached)

return 1; (the greatest common divisor of (7,5) is 1)

Example 3: Fibonacci sequence:

This Fibonacci sequence appeared the first time in the book entitled Liber Abaci (1202) written by Leonardo of Pisa (known as Fibonacci). The sequence is derived from the growth of an unrealistic rabbit population. In the growth, the rabbits are assumed to be immortal. The origin problem asked the number of pair of rabbits after a year with these following conditions:

- Having a pair of rabbits (one male and one female) from the beginning.
- The pair of rabbits will mate at the age of one month.
- After having mated one month, one pair of rabbits always produces a pair of rabbits (one male and one female) every month.

This is explaining how the process is:

- At the end of the first month, the original rabbits are one month age and can mate, but no new pair of rabbits is produced.
- At the end of the second month, a new pair of rabbits is produced. However, the new pair of rabbits cannot mate until they get one month

age. Therefore, now there are two pairs of rabbits but just one can produce new rabbits in the next month.

- At the end of the third month, one more new pair of rabbits is produced by the original ones, and the second pair of rabbits now mate. Therefore, there are three pairs of rabbits now, and two of them can produce new rabbits in the next month.
- At the end of the fourth month, two new pair of rabbits are produced by two mated couples of rabbits, and the ones born in the third month now mate. Therefore, now, five pairs of rabbits are available and three of them can produce new rabbits in the next month.
- Among the pairs of rabbits existed from the $(n-1)^{th}$ month, the ones can produce new rabbits are the ones alive from the $(n - 2)^{th}$ month. Therefore, the pairs of rabbits in the nth month are equal to the sum of the pairs of rabbits from the $(n-1)^{th}$ and $(n-2)^{th}$ month.

“The name "Fibonacci sequence" was first used by the 19th-century number theorist Édouard Lucas.”[16]

This following algorithm will show how to find the nth number in the Fibonacci sequence. The function will get the value of the nth number in the Fibonacci sequence. The natural number n is passed as parameters to the function.

```
public static int getFibonacinth(int n){
    if (n==1 || n==2) return 1;
    else return getFibonacinth(n-1)+getFibonacinth(n-2);
}
```

This getFibonacinth function will set the anchor step for the first and the second Fibonacci numbers (both of them take the same value which is one). The recursive step means that in order to calculate any Fibonacci number, we need to add the right previous two other Fibonacci numbers. If any of those previous Fibonacci numbers are missing, the same process is applied to find them until the anchor step is reached.

For example, the function can be used to calculate the fifth(5th) Fibonacci number as follows:

getFibonacinth(5)

= getFibonacinth(4) + getFibonacinth(3)

= getFibonacinth(3) + getFibonacinth(2) + getFibonacinth(3)

= getFibonacinth(2) + getFibonacinth(1) + getFibonacinth(2) + getFibonacinth(2) + getFibonacinth(1)

(the anchor step is reached)

= 1 + 1 + 1 + 1 + 1

return 5; (the fifth(5th) Fibonacci number is 5 (1 1 2 3 5))

1.7. Recursive data types

Recursive data types (or inductive data types) are important types in programming. In a recursive type, one value can contain the same type value(s) such as binary trees in which node(s) can contain node(s).

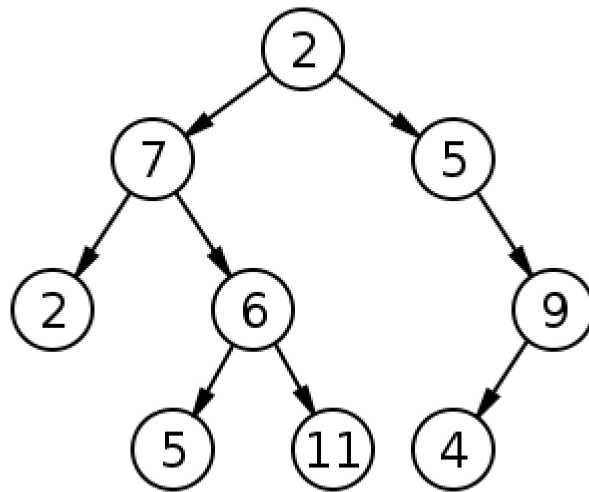


Image 2: A binary tree [17]

Recursive data types are flexible. The ability of dynamically growing their size gives a good response to runtime requirements instead of having to be set at compile time in case of a static array's size.

1.8. Comparison between iterative and recursive methods

The comparison between recursion and iteration is a classic topic. Both of them are fundamental and widely used to solve problems. This section covers the comparison between those.

Both recursion and iteration have the same ideas of solving problem by breaking down the original problem into the simpler instances which can be solved directly and easily. However, they are quite different from each other.

By using iteration, the solution to the problem will be changed to a chain of small steps. From the beginning to the end of the chain, each step will occur once and one by one. For example, if we want to find a total of five numbers using iteration, we can make a chain of five steps of adding (one number for each step). The process is illustrated here:

The first step: adding the first number. Done! Go to next step!

The second step: adding the second number. Done! Go to next step!

The third step: adding the third number. Done! Go to next step!

The fourth step: adding the fourth number. Done! Go to next step!

The fourth step: adding the fifth number. Done! Go to next step!

From the illustration, we can see the steps have been done clearly. In the iteration, we know the step we are currently in and the number should be added in that step. Therefore, the iteration can be considered to be easier to understand than recursion which is not quite clear.

By using recursion, the whole process of solving a type of problem will be repeatedly occurring on the subproblems which are derived from the original problem until the result is found. For instance, we take the example of factorial again. To calculate n! by recursion, we need to know (n-1)! to use the formula: n! = n x (n-1)! (see again example 3 in part 1.1.5). And the process to calculate (n-1)! is the same process to calculate n!. The process will repeat until n = 0 where the problem is simple enough to solve directly (0! = 1). Through that, we can see that recursion is not clear for each step. On the other hand, the whole process is always repeated except for the subproblems that can be solved directly.

Although, these methods are different from each other, they are both equally important. It is necessary to understand them thoroughly and apply them well because they appear in almost all computer algorithms existing.[18]

Even though it seems easier to understand and apply iteration than recursion from the beginning, recursion can describe things in a much simpler way than iteration in very complex and abstract problems. In addition, both recursion and iteration can reach the same expressive power. Therefore, the choice of using either of them often depends more on the preference of the programmer than on the nature of the problem.[19]

II. EXAMPLE OF USING RECURSION

1. Backtracking algorithm

1.1. Introduction

The backtracking algorithm is the method to find the solution by trying every possibility one by one. That can be described as a process of deeply searching the solution(s) in a set of possibilities. During that process, if we notice that we are in a wrong direction to solve the problem, we will go back to the nearest point at which we can have other directions to choose. Then, we choose the next direction. After all the directions from that point are chosen, we will go back to the previous point and continue to try other possibilities. The process will stop when there is no other choice. [20]

1.2. The Eight Queens Puzzle

Description:

The eight queens puzzle requires to have eight queens placed on a chess board (with 8 rows and 8 columns) with the condition that one queen cannot attack the others.

Note: A queen will attack another queen if both of them are in the same row, column or diagonal.

Function and explanation

```
public void trying(int j){
     for(int i=0; i<8;++i){
               if (board[i][j] == 0){
                    placeQueen(i,j);
                    if (j==7) printOutPut();
                    else trying(j+1);
                    removeQueen(j+1);
               }
     }
}
```

This trying function will place the queens on the board by the following process: Try to place the queen in the row j from the first column. If there is an empty cell then the do placeQueen function which will check if the cell is suitable to place a queen. Then we place the next queen. We keep trying to place queens to the board until we cannot place (due to the fact that there is no other suitable cell or there is no queen). Then we go back to the point that we have choices to place the queens. Then we choose another choice than the direction we just went in.

2. Divide and Conquer algorithm

2.1. Introduction

The divide and conquer algorithm is an important algorithm in computer science. The divide and conquer algorithm is derived from multi-branched recursion. The idea of applying the divide and conquer algorithm is to break an original problem into two or more subproblems in the same or similar categories recursively. The subproblems will be solved directly when they are simple enough. Then the combination of all the subproblems' results will give the final answer to the original problem.

Divide and conquer is the foundation of efficient algorithms to solve all all kind of problems, for example, sorting with quicksort and merge sort algorithms or multiplying large numbers with the Karatsuba algorithm.

Using the divide and conquer algorithm is a very useful and important technique. However, applying the algorithm usually requires changing the original problem to be more general and complicated to apply recursive process. The most difficult thing is that there is no systematic method to find the right way to change the original problem. That task requires a lot of experience in solving problems from time to time.

The name “divide and conquer” is considered to be replaced by “decrease and conquer” in case the original problem has only one subproblem.

Mathematical induction is usually the method to prove the correctness of a divide and conquer algorithm.[21]

2.2. Find the maximum

Description

An unordered sequence of natural numbers (positive integers) is given. Find the maximum value in that sequence.

Functions and explaination

```
public class FindMax {

	int mid;

	int maxFirst, maxSecond;

	public int findMaxNumber(int[] sequence, int left, int
right) {

		if (left == right) return sequence[left];

		else {

		mid = (left+right)/2;

		maxFirst = findMaxNumber(sequence, left, mid);

		maxSecond = findMaxNumber(sequence,mid+1,right);

		if(maxFirst>maxSecond) return maxFirst;

		else return maxSecond;

	}

}
```

This findMaxNumber function does the following tasks in order to find the maximum:

If the first index and the last index are equal, which means that just one element exists (?)in the sequence, the function returns the max equal to that only element.
Otherwise, the sequence is divided into two subsequences, and the find max algorithm is applied on both of the subsequences to find out two maximum (one for each subsequence).
Finally, comparing the two maximums, the greater one is the final maximum.

3. Dynamic programming

3.1 Introduction

The recursive algorithm has the advantage of being easy to install. However, due to the character of recursion, the process, which is provided by the algorithm, usually entails a huge amount of calculation with great demands on memory space. Dynamic programming solves complex problems by breaking them down into smaller and simpler sub problems. Next, those sub problems are going to be solved, and the received results are stored to be reused later for a lager instances. Then, all the results collected are combined to reach the final solution. By using the stored results, this method saves a lot of time compared to other naive recursive methods.[22]

The skill of using dynamic programming to solve problem takes a lot of time to master. The reason why the process of finding a suitable dynamic programming algorithm for problems is difficult is because there is no general algorithm that can resolve all the dynamic programming problems.

3.2. The longest increasing subsequence

Description

The problem of finding the longest increasing subsequence is stated as follows: A sequence of integer numbers given in this problem contains n elements. The task is finding an increasing sequence which includes some or all elements from the original sequence and keeping the original order, and it must be the longest one that could be found. Note that the elements of the new sequence are not necessarily consecutive elements in the initial sequence.[23]

Function and explanation

```
public void search(){

    for(int i=0; i<a; ++i){
        B[i] = 1; C[i] = i;
        for(int j=0; j<i; ++j){
            if(B[j]+1>B[i]) {
            B[i] = B[j]+1;
            C[i] = j;
            }
        }
}
```

To determine how many elements are contained in the longest subsequence of a given subsequence, this search function performs the following tasks:
It creates an array to store the counter (B[i]). Then it starts the counter from one at a specific element. Next, the function compare will then compare that element with all other element after it. Whenever a greater element appears, the starting point will change to that element and the counter increases by one. The process continues until the sequence ends. Finally comparing all the counters, the greatest value is the length of the longest subsequence.

III. PRACTICAL APPLICATION

This section describes my practical report of recursion application. The purpose of this report is to reveal the full potential of recursion, by doing and examining the classical algorithm – quicksort algorithm (one of the quickest sorting algorithms existed).

1. Introduction of the Scala programming language

Scala was invented by Ph.D. Martin Odersky in 2001 with the purpose of delivering more concise, beautiful and type-safe expressions in programming. Scala is designed to be a good combination of object-oriented and functional languages' features. Therefore, Scala can use object data structures similar to, for example, Java while the number of codes can be reduced to a half or one third of an equivalent program written by Java. In addition to the combination, Scala has many good features such as:

- **Seamless integration with Java**

Scala uses Java Virtual Machine to run programs on the Java VM. Moreover, Scala has a compatible byte code with Java, thus the existing Java application codes and libraries can be ultimately used in Scala. The integration here is seamless. Therefore, Java and Scala can call each other. Besides, the familiar Java development tools such as Eclipse, NetBeans or Intellij also support Scala. If we are familiar with Java, it does not take a long time to become a proficient user of Scala. [24]

- **Static**

"Scala is equipped with an expressive type system that enforces statically that abstractions are used in a safe and coherent manner. In particular, the type system supports:

- generic classes,
- variance annotations,

- upper and lower type bounds,
- inner classes and abstract types as object members,
- compound types,
- explicitly typed self references,
- views, and
- polymorphic methods.

A local type inference mechanism ensures that the user is not required to annotate the program with redundant type information. In combination, these features provide a powerful basis for the safe reuse of programming abstractions and for the type-safe extension of software." [25]

- **Extensible**

"In practice, the development of domain-specific applications often requires domain-specific language extensions. Scala provides a unique combination of language mechanisms that make it easy to smoothly add new language constructs in form of libraries:

- any method may be used as an infix or postfix operator, and
- closures are constructed automatically depending on the expected type (target typing).

A joint use of both features facilitates the definition of new statements without extending the syntax and without using macro-like meta-programming facilities."[26]

- **Cooperates with Java and .NET**

Scala cooperates well with the popular Java 2 Runtime Environment (JRE), has the same compilation model (separate compilation, dynamic class loading) as Java, allows access to thousands of existing high-quality libraries, and supports the .NET Framework Common Language Runtime (CLR).[27]

- **Highly reliable compiler**

After some effort in studying new syntax, anyone can write programs with Scala as well as Java and can enjoy their lives as programmers with all the features mentioned above and even more than that.

Despite the fact that Scala is a new developed programming language, many existing companies are now changing to Scala to improve “their development productivity, applications scalability and overall reliability.” For instance, Robey Pointer moved the Twitter's Ruby core message queue to Scala core message queue. This change occurred because of the need of scalability of the company's operation to satisfy the fast growing community on Twitter. The Tweet rates were already at the number of 5000 per minute during the Obama Inauguration.[28]

2. Quicksort algorithm

2.1 Introduction

Quicksort is one of the quickest sorting algorithms whose basic form was introduced for the first time in 1960 by C. A. R. Hoare. After that, the quicksort algorithm has been widely studied. The features that make the quicksort algorithm popular are:

Relatively easy to implement.

Working well with a wide range of kinds of input data.

Fewer resources required in many situations compared to other sorting methods.[29]

The idea behind the quicksort algorithm can be described concisely as follows:

If there is less than two elements in a need-to-be-sorted array, return the array immediately because one or no element in an array means that the array is already sorted.

Otherwise, we choose a middle element to be a pivot. Then we divide the array into three sub-arrays:

- ✓ The array contains all the elements that are less than the pivot
- ✓ The array contains all the elements that are greater the pivot
- ✓ The pivot itself.

Finally, we sort the first and the second sub-array by using the same quicksort algorithm and concatenate them in order.[30]

In the following sections, 2.2 and 2.3, the quicksort algorithm is implemented by two different styles of programming which are imperative style and functioning style. By comparing the amount of code lines, we can see that using recursion with functioning programming style probably can save us a lot of time on code work.

2.2 Quicksort function written in imperative programming style

```
object quicksortI {
          private val sequence = Array[Int]()
          private val lengthoS = 0

          def sort(sequence: Array[Int]){
                    var this.sequence = sequence
                    var lengthoS = sequence.length
                    quicksort(0, lengthoS-1)
          }

          def swap(e1: Int, e2:Int){
            val temp = sequence(e1)
            sequence(e1) = sequence(e2)
            sequence(e2) = temp
          }

          def quicksort(left:Int, right:Int){
            val pivot = sequence((left+right)/2)
```

```
              var el = left
              var er = right

              while (el<=er) {
                while(sequence(el)<pivot) {el+=1}
                while(sequence(er)<pivot) {er-=1}
                if(el<=er){
                  swap(el,er)
                  el+=1
                  er-=1
                }
              }

              if(left<er) quicksort(left, er)
              if (el<right) quicksort(el,right)
            }

}
```

2.3 Quicksort function written in functional programming style

```
object quicksortF {
            def quicksort(sequence:Array[Int]):Array[Int] = {
              if(sequence.length<2) sequence
              else{
                val pivot = sequence(sequence.length/2)
                Array.concat(
                      quicksort (sequence filter (pivot>_)),
                      sequence filter (pivot==_),
                      quicksort(sequence filter (pivot<_)))
              }
            }
```

```
}
```

IV. CONCLUSION

Recursion has its very important role in computer sciences. Recursion is not only a basic concept to build another algorithm but also a way to describe an algorithm more beautifully and concisely. Using the recursive method suitably can achieve the same results as other methods but in a more beautiful manner and with less amount of code work. Then I personally recommend everyone who studying algorithm to study recursion and all its developed branches carefully. That brings programmers and coders many benefits.

REFERENCES

[1] Oxford University Press (2011), *Oxford Dictionaries*, access day 17/11/2011, http://oxforddictionaries.com

[2] Oxford University Press (2011), *Oxford Dictionaries*, access day 17/11/2011, http://oxforddictionaries.com/definition/computer%20science

[3] Glenn BrookShear, J. (1994), *Computer Science An Overview* (4th ed, pp. 1). Redwood City: The Benjamin/Cummings Publishing Company, Inc.

[4] Glenn BrookShear, J. (1994), *Computer Science An Overview* (4th ed, pp. 1). Redwood City: The Benjamin/Cummings Publishing Company, Inc.

[5] Oxford University Press (2011), *Oxford Dictionaries*, access day 17/11/2011, http://oxforddictionaries.com/definition/algorithm.

[6] Wikipedia, the free encyclopedia, *Mathematical induction*, access day, 15/12/2011, http://en.wikipedia.org/wiki/Mathematical_induction access day

[7] Rosen, Kenneth H. (2007), *Discrete Mathematics and Its Applications* (6th ed, pp. 265). New York: The McGraw-Hill Companies, Inc.

[8] Rosen, Kenneth H. (2007), *Discrete Mathematics and Its Applications* (6th ed, pp. 264). New York: The McGraw-Hill Companies, Inc.

[9] Wikipedia, the free encyclopedia, *File:Domino effect.jpg*, access day 24/10/2011, http://en.wikipedia.org/wiki/File:Domino_effect.jpg

[10] Rosen, Kenneth H. (2007), *Discrete Mathematics and Its Applications* (6th ed, pp. 294). New York: The McGraw-Hill Companies, Inc.

[11] Wikipedia, the free encyclopedia, *Recursion*, access day 20/6/2011 http://en.wikipedia.org/wiki/Recursion

[12] Wikipedia, the free encyclopedia, *Factorial*, access day 3/1/2012, http://en.wikipedia.org/wiki/Factorial#Applications

[13] Robert Sedgewick, *Algorithms in C++*, (3rd ed, part 1-4, pp. 203). Boston: Addison-Wesley Publishing Company, Inc.

[14] Donald Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd edition (1981), p. 318.

[15] Robert Sedgewick, *Algorithms in C++*, (3rd ed, part 1-4, pp. 205). Boston: Addison-Wesley Publishing Company, Inc.

[16] Wikipedia, the free encyclopedia, *Fibonacci number*, access day 3/1/2012, http://en.wikipedia.org/wiki/Fibonacci_number#Origins

[17] Wikipedia, the free encyclopedia, *File:Binary tree.svg*, access day 17/01/2012, http://en.wikipedia.org/wiki/File:Binary_tree.svg

[18] AI Horizon, *Recursion and Iteration*, access day 4/1/2012 http://www.aihorizon.com/essays/basiccs/general/recit.html

[19] Maurizio Gabbrielli, Simone Martini, *Programming Languages: Principles and Paradigms*, (pp. 159). London: Springer-Verlag London Limited 2010

[20] Wikipedia, the free encyclopedia, *Quay lui (khoa học máy tính),* access day 17/01/2012, http://vi.wikipedia.org/wiki/Quay_lui

[21] Wikipedia, the free encyclopedia, *Divide and conquer algorithm*, access day 14/12/2011, http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm

[22] Wikipedia, the free encyclopedia, *Dynamic programming,* access day 19/12/2011 http://en.wikipedia.org/wiki/Dynamic_programming

[23] Lớp 12a5 - Tứ Kỳ Hải Dương, *Bài toán dãy con tăng – giảm dài nhất*, access day 22/11/2011, http://lop12a5thpttk-0609.forumotion.net/t244-topic

[24] Admin of Scala-lang.org website, The Scala Programming Language, access day 5/12/2011, http://www.scala-lang.org/node/25

[25] Admin of Scala-lang.org website, The Scala Programming Language, access day 5/12/2011, http://www.scala-lang.org/node/25

[26] Admin of Scala-lang.org website, The Scala Programming Language, access day 5/12/2011, http://www.scala-lang.org/node/25

[27] Admin of Scala-lang.org website, The Scala Programming Language, access day 5/12/2011, http://www.scala-lang.org/node/25

[28] Admin of Scala-lang.org website, The Scala Programming Language, access day 5/12/2011, http://www.scala-lang.org/node/25

[29] Robert Sedgewick, *Algorithms in C++*, (3rd ed, part 1-4, pp. 315). Boston: Addison-Wesley Publishing Company, Inc.

[30] Martin Odersky, *Scala by example*, (draft May 24, 2011, pp. 4). Switzeland: programming methods laboratory

V. APPENDIX

Source code in Java and screen shots:

The eight queens puzzle

```
public class EightQueens {

        //variables
        int[][] board = new int[8][8];
        int cach = 1;

        public void printOutPut(){
                if(cach<4){
                System.out.println("Begin printOutPut: ");
                System.out.println("Solution Sample " + cach + " :
");
                for(int i=0;i<8;++i){
                        for(int j=0;j<8;++j){
                                if (board[i][j]<0){
                                        System.out.print("
x ");
                                }
                                else
                                        System.out.print("
" + board[i][j] + " ");
                        }
                        System.out.println();
                }
                ++cach;
                }
        }

        public void placeQueen(int i, int j){
                // duong cheo va duong thang
                for(int x = 0; x<8; ++x){
                        for(int y = 0; y<8; ++y){
```

```
                                        if(( ((x+y) == (i+j))
                                                        || (x-
y)==(i-j)
            || (x==i)
                        || (y==j)) && board[x][y]==0){
                                                board[x][y] = -
(j+1);
                                        }
                                }
                        }
                        board[i][j] = j+1;
            }

            public void removeQueen(int id){
                        for(int x = 0; x<8; ++x){
                                for(int y = 0; y<8; ++y){
                                        if(abs(board[x][y])==id)
board[x][y]= 0;
                                }
                        }
            }

            public int abs(int i){
                        if(i<0) i=i*(-1);
                        return i;
            }

            public void trying(int j){
                        for(int i=0; i<8;++i){
                                if (board[i][j] == 0){
                                        placeQueen(i,j);
                                        if (j==7) printOutPut();
                                        else {
                                                trying(j+1);
                                        }
                                        removeQueen(j+1);
```

```
                        }
                }
        }

}

public class EightQueens {

        //variables
        int[][] board = new int[8][8];
        int cach = 1;

        public void printOutPut(){
                if(cach<4){
                System.out.println("Begin printOutPut: ");
                System.out.println("Solution Sample " + cach + " :
");
                for(int i=0;i<8;++i){
                        for(int j=0;j<8;++j){
                                if (board[i][j]<0){
                                        System.out.print("
x ");
                                }
                                else
                                        System.out.print("
" + board[i][j] + " ");
                        }
                        System.out.println();
                }
                ++cach;
                }
        }

        public void placeQueen(int i, int j){
                // duong cheo va duong thang
                for(int x = 0; x<8; ++x){
                        for(int y = 0; y<8; ++y){
                                if(( ((x+y) == (i+j))
```

```
                                                                        || (x-
y)==(i-j)
            || (x==i)
                        || (y==j)) && board[x][y]==0){
                                                                board[x][y] = -
(j+1);
                                                    }
                                        }
                        }
                        board[i][j] = j+1;
            }

            public void removeQueen(int id){
                        for(int x = 0; x<8; ++x){
                                    for(int y = 0; y<8; ++y){
                                                if(abs(board[x][y])==id)
board[x][y]= 0;
                                    }
                        }
            }

            public int abs(int i){
                        if(i<0) i=i*(-1);
                        return i;
            }

            public void trying(int j){
                        for(int i=0; i<8;++i){
                                    if (board[i][j] == 0){
                                                placeQueen(i,j);
                                                if (j==7) printOutPut();
                                                else {
                                                            trying(j+1);
                                                }
                                                removeQueen(j+1);
                                    }
```

```
                    }
          }

}
```

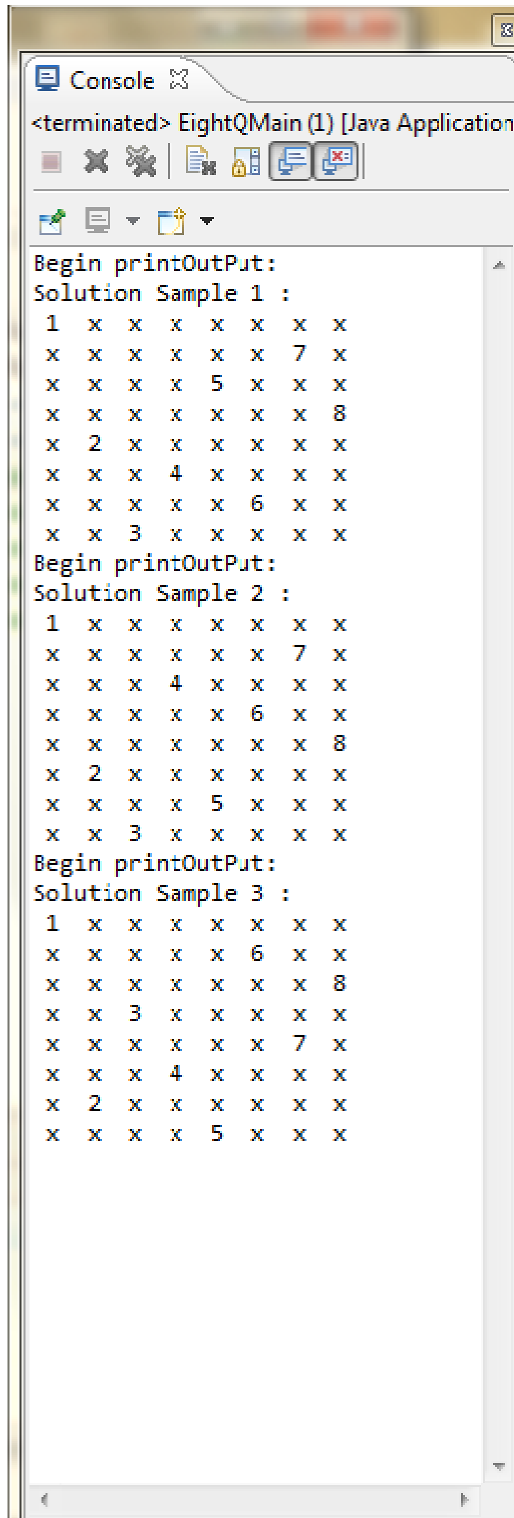


Figure 1: The screenshot of three sample results of the eight queen puzzle program.

Find Maximum program

```
public class FindMax {

            public int findMaxNumber(int[] sequence, int left, int right){
                        System.out.println("left: " + left+" " + "right: " + 
right );
                        if (left == right) return sequence[left];
                        else {
                                    int mid = (left+right)/2;
                                    System.out.println("mid: " + mid);

                                    int maxFirst = findMaxNumber(sequence, 
left, mid);
                                    System.out.println("Max of first half: " 
+ maxFirst);
                                    int maxSecond = 
findMaxNumber(sequence,mid+1,right);
                                    System.out.println("Max of second half: " 
+ maxSecond);

                                    if(maxFirst>maxSecond) {

            System.out.println("maxFirst>maxSecond");
                                                System.out.println("The 
current max is: " + maxFirst);
                                                System.out.println("end");
                                                System.out.println();
                                                return maxFirst;
                                    }
                                    else {

            System.out.println("maxFirst<maxSecond");
                                                System.out.println("The 
current max is: " + maxSecond);
                                                System.out.println("end");
                                                System.out.println();
                                                return maxSecond;
```

```
                        }
                }
        }

}


public class FindMax {

        public int findMaxNumber(int[] sequence, int left, int right){
                System.out.println("left: " + left+" " + "right: " +
right );
                if (left == right) return sequence[left];
                else {
                        int mid = (left+right)/2;
                        System.out.println("mid: " + mid);

                        int maxFirst = findMaxNumber(sequence,
left, mid);
                        System.out.println("Max of first half: "
+ maxFirst);
                        int maxSecond =
findMaxNumber(sequence,mid+1,right);
                        System.out.println("Max of second half: "
+ maxSecond);

                        if(maxFirst>maxSecond) {

        System.out.println("maxFirst>maxSecond");
                                System.out.println("The
current max is: " + maxFirst);
                                System.out.println("end");
                                System.out.println();
                                return maxFirst;
                        }
                        else {

        System.out.println("maxFirst<maxSecond");
```

```
                                        System.out.println("The 
current max is: " + maxSecond);
                                        System.out.println("end");
                                        System.out.println();
                                        return maxSecond;
                              }
                     }
          }

}
```

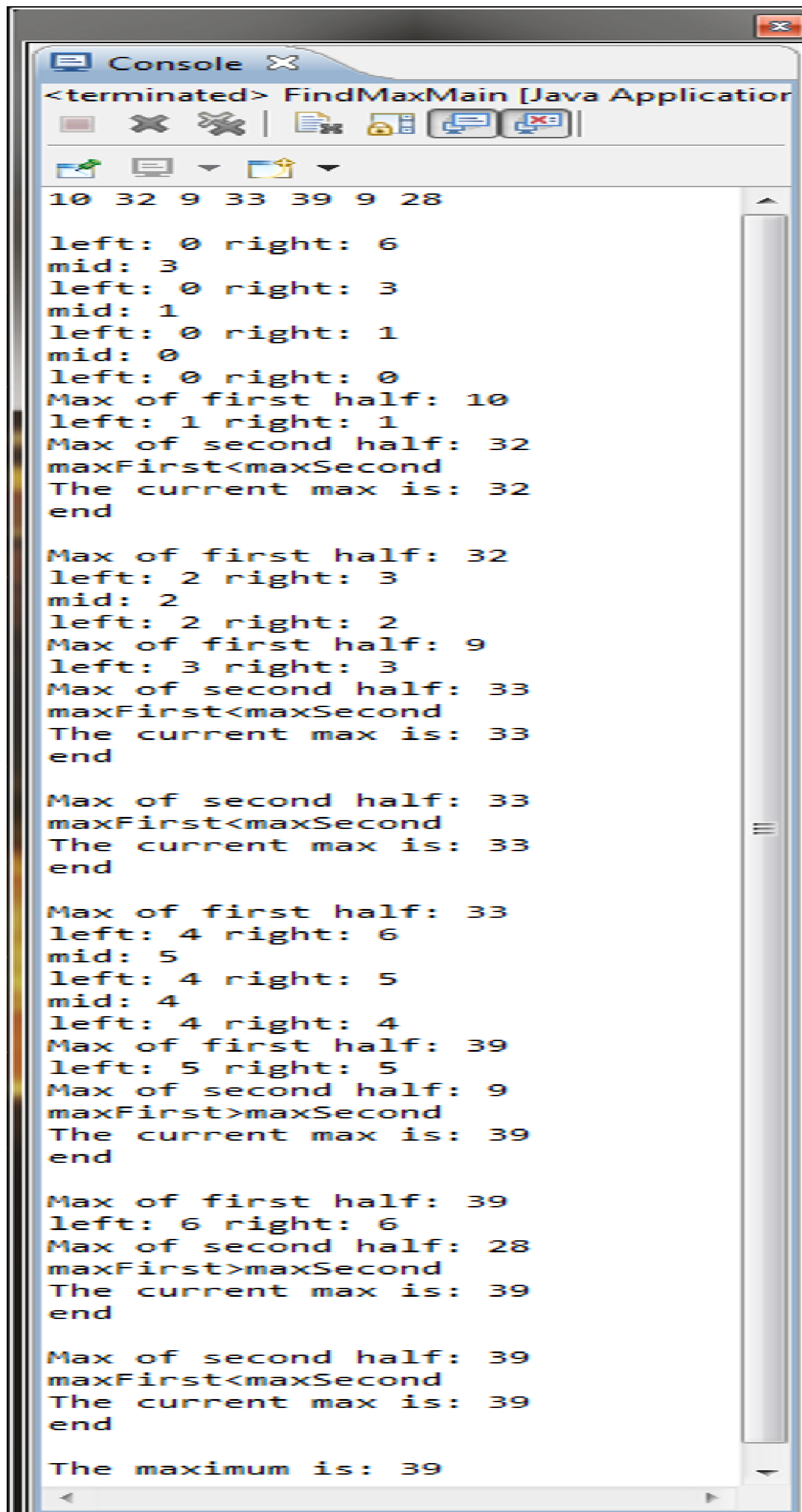


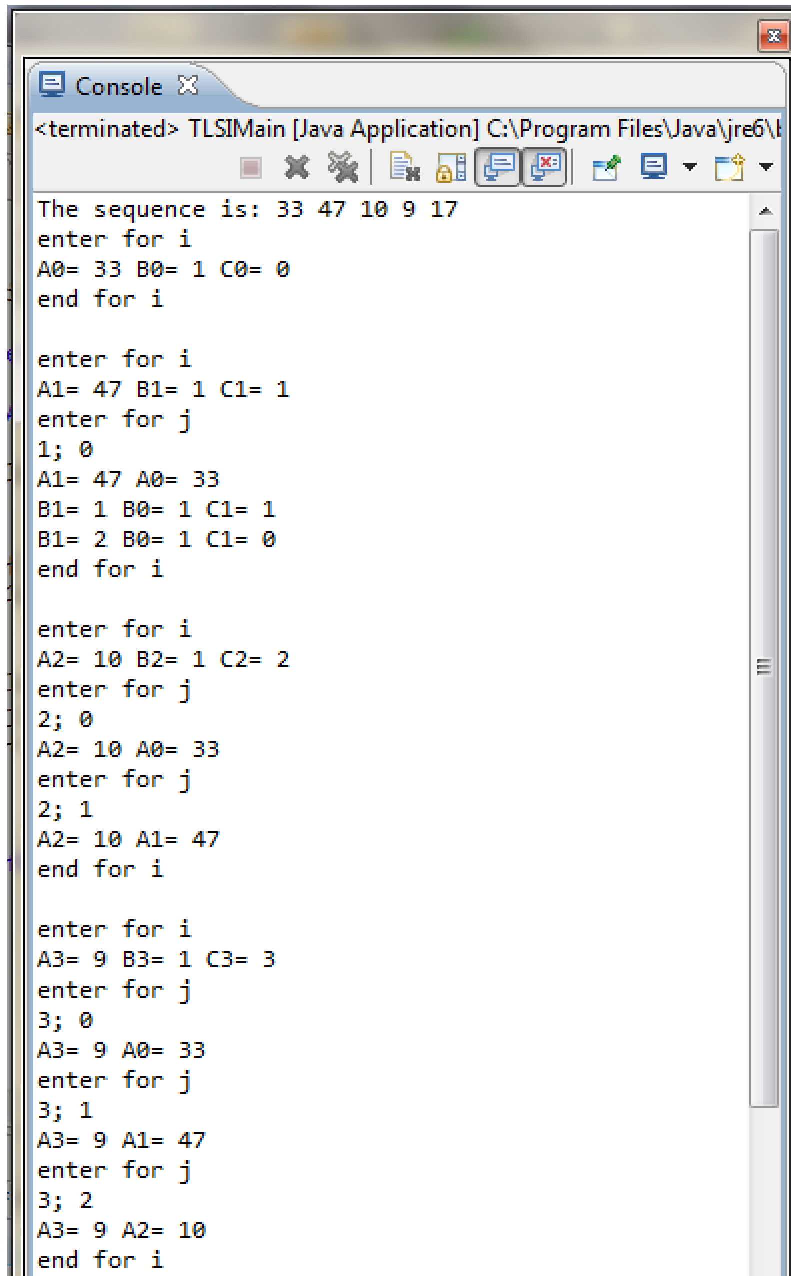
Figure 2: This is the screenshot of one sample result of the find maximum program

The longest increasing subsequence

```
import java.util.Random;
import java.util.Scanner;

public class TLISubSequence {

	//variables
	Scanner scan = new Scanner(System.in);
	int a;
	int[] A, B, C;

	public TLISubSequence(){

	}

	public void inputEntered(){
		Random r = new Random();
		a=5;
		A = new int[a];
		System.out.print("The sequence is: ");
		for(int i=0; i<a;++i){
			A[i] = r.nextInt(50);
			System.out.print(A[i] + " ");
		}
		System.out.println();
		B = new int[a];
		C = new int[a];
	}

	public void printOutPut(){
		int max = B[0];
		for(int i=1; i<a; ++i){
			if(B[i]>max){
				max = B[i];
			}
		}
```

```
                        System.out.println("The longest increasing 
subsequence has: " + max +" numbers.");
            }

            public void search(){
                        for(int i=0; i<a; ++i){
                                    //System.out.println("enter for i");
                                    B[i] = 1;
                                    C[i] = i;
                                    /*System.out.println("A" + i + "= " +A[i] 
+   " "
                                                            + "B" + i + "= " + 
B[i] + " "

            + "C" + i + "= " + C[i]);*/
                                    for(int j=0; j<i; ++j){
                                                /*System.out.println("enter 
for j");
                                                System.out.println(i + "; " + 
j);
                                                System.out.println("A" + i + 
"= " + A[i] + " " + "A" + j + "= " + A[j]);*/
                                                if(A[j]<A[i]){

            /*System.out.println("B" + i + "= " +B[i] +   " "

                                                            + "B" + j + "= " + 
B[j] + " "


            + "C" + i + "= " + C[i]);*/
                                                            if(B[j]+1>B[i]) {
                                                                        B[i] = 
B[j]+1;
                                                                        C[i] = 
j;
                                                            }
```

```
                    /*System.out.println("B" + i + "= " +B[i] +  " "

                    + "B" + j + "= " + B[j] + " "

                                            + "C" + i + "= " + C[i]);*/
                                                        }
                                            }

                                            System.out.println("end for i");
                                            System.out.println();
                                }
                    }


}


import java.util.Random;

public class TLSIMain {

            public static void main(String[] args){
                        TLISubSequence tl = new TLISubSequence();
                        tl.inputEntered();
                        tl.search();
                        tl.printOutPut();
            }

}
```


Console
<terminated> TLSIMain [Java Application] C:\Program Files\Java\jre6\
The sequence is: 33 47 10 9 17
enter for i
A0= 33 B0= 1 C0= 0
end for i
enter for i
A1= 47 B1= 1 C1= 1
enter for j
1; 0
A1= 47 A0= 33
B1= 1 B0= 1 C1= 1
B1= 2 B0= 1 C1= 0
end for i
enter for i
A2= 10 B2= 1 C2= 2
enter for j
2; 0
A2= 10 A0= 33
enter for j
2; 1
A2= 10 A1= 47
end for i
enter for i
A3= 9 B3= 1 C3= 3
enter for j
3; 0
A3= 9 A0= 33
enter for j
3; 1
A3= 9 A1= 47
enter for j
3; 2
A3= 9 A2= 10
end for i

```
enter for i
A4= 17 B4= 1 C4= 4
enter for j
4; 0
A4= 17 A0= 33
enter for j
4; 1
A4= 17 A1= 47
enter for j
4; 2
A4= 17 A2= 10
B4= 1 B2= 1 C4= 4
B4= 2 B2= 1 C4= 2
enter for j
4; 3
A4= 17 A3= 9
B4= 2 B3= 1 C4= 2
B4= 2 B3= 1 C4= 2
end for i

The longest increasing supsequence has: 2 numbers.
```

Figure 3: These screenshots show the process of finding the longest increasing subsequence using the written program.