Czech university of life sciences Prague

Faculty of economics and management

Department of information technologies

Bachelor thesis

Data structures and efficient algorithm design in java

By Jean Bertrand Habinshuti

©2018 CULS Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Jean Bertrand Habinshuti

Informatics

Thesis title

Data structures and efficient algorithm design in Java

Objectives of thesis

The main goal of the thesis is to analyze algorithm performance in terms of memory uase and running time.

It will start by estimating algorithms efficiency mathematically, then different tasks in computer environment will be analyzed such as sorting, searching and the comparison will be provided based on their common growth functions.

Although any programming language may be used , here it will be used Java programming language to implements algorithms. The results should still be the same since the running time and memory taken by a program depend on how the algorithm is designed rather than the programming language used. That being said, anyone will be able to apply same methods used in this thesis using any other programming language such as C# or C++.

Methodology

At the beginning will be algorithms visualized. There are many methods to visualize algorithms, in this thesis I will be using the pseudo code written using Java language syntax and charts where necessary.

Then it will be created a simple web site for user to better understanding the end result of a carefully designed algorithms. They will be able to perform different tasks such as add element into the tree,search for an element in the tree, delete element from the tree.

The thesis will be written using a document preparation system(latex).

The aim of the survey part will be to investigate how often programmers care about the performance of programs in their everyday design tasks or whether people should bother selecting design methods at all.

The proposed extent of the thesis

30-40 pages

Keywords

Algorithms, data structures, java, java ee, tree, binary, fibonacci, sorting,Efficiency,prime numbers,Big O notation

---

Recommended information sources

Data structures and algorithms analysis in java , by mark allen weiss, 3rd edition .Florida international univesity,ISBN-13: 9780132576277

D. E. Knuth, The Art of Computer Programming: Vol. 3: Sorting and Searching, 2d ed., Addison-Wesley, Reading, Mass, 1998,ISBN-13: 9780201896855

Introduction to java programming , 8th edition , by Y.Daniel Liang. Armstrong atlantic state university,ISBN-13: 978-0-13-213080-6

Java ee 6 development within netbeans 7 by David R.heffelnger, first published 2011, ISBN 978-1-849512-70-1

Expected date of thesis defence

2017/18 SS – FEM

The Bachelor Thesis Supervisor

Ing. Martin Havránek, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 30. 11. 2017

Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 30. 11. 2017

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 28. 02. 2018

DECLARATION

I declare,that i have independently worked on this bachelor thesis titled"Data structures and efficient algorithm design in java" under the guidance of the thesis supervisor and using the specialized literature and other information sources that are quoted in the thesis and listed at the end of the thesis. As the author of this bachelor thesis, I further declare that I did not infringe the copyrights of third parties in connection with its creation.

Done in Prague on 28 February 2018

Aknowledgment

I would like to thank my thesis supervisor Ing. Martin Havránek,Ph.D for being so cooperative by providing me the guidance and advice that was necessary throughout the development of this thesis.I would also like to take this opportunity to thank everyone from my family in particularly my beautiful mother who, despite not being academic did everything she could in her power so that i could have a good education.I would also like to show my gratitude to the Czech government for granting me the opportunity to come to study in one of the best universities through its cooperation with the Rwandan government.I also place on record,my sense of gratitude to the European union for giving me the opportunity to spend one year at Plymouth University in United Kingdom through the Erasmus program.Finally,completing this work would have been difficult were it not for the support provided by friends,classmates ,lecturers and members from the faculty of economics and management.

Data structures and efficient algorithm design in java

Abstract

The number of applications users has increased drastically during past years and this lead to high demand for systems to be designed efficiently in order to cope with this increase.Throughout this thesis,i have demonstrated how algorithm design can affect performance.Different approaches to solving common tasks in computer science such as searching element from a long list of elements are analyzed and based on the outcome, the best approach is selected.The implementation is done using java programming language.At the end of this thesis,a simple webpage showing the binary search tree is created.The purpose of the webpage is to allow users to see graphically how data can be arranged in such way that operations on them can be done faster. The webpage url can be found here https://bstviz.000webhostapp.com.A survey was conducted with objective to get the respondent's views on the role of focusing on efficiency as a way of dealing with high increase in application users and data volume. The results of the survey can be found in the survey part of this thesis and in the appendix

Keywords:Algorithms,data structures,binary search tree,binary search,Big O notation,java,function,complexity function,linear search,binary search,efficiency,running time.

Datové struktury a efektivní návrh algoritmu v javě

Abstrakt

Počet uživatelů aplikací se v posledních letech drasticky zvýšil a to vedlo k vysoké poptávce po efektivním navrhování systémů, aby bylo možné zvládnout toto zvýšení. V této práci jsem prokázal, jak může algoritmový design ovlivnit výkon. Různé přístupy k řešení běžných úkoly v oblasti počítačové vědy, jako je vyhledávací prvek z dlouhého seznamu prvků, jsou analyzovány a na základě výsledku je zvolen nejlepší přístup. Implementace se provádí pomocí programovacího java jazyka. Na konci této práce je zobrazena jednoduchá stránka s binárním je vytvořen vyhledávací strom. Účelem webové stránky je umožnit uživatelům graficky vidět, jak mohou být data uspořádána tak, aby operace na nich byla rychlejší. Adresa URL webové stránky naleznete zde https: //b-stviz.000webhostapp.com.Byly provedeny průzkumy s cílem získat názory respondenta na úlohu zaměřit se na efektivitu jako způsob řešení vysokého nárůstu počtu uživatelů aplikací a objemu dat. Výsledky průzkumu lze nalézt v části průzkumu této práce a v dodatku

klíčová slova:Algoritmy, datové struktury, binární vyhledávací strom, binární vyhledávání, velká O notace, java, funkce, funkce složitosti, lineární hledání, binární vyhledávání, efektivita, doba běhu

Contents

1 Introduction **10**

2 Thesis objectives and methodology **11**

3 Data structures **12**
3.1 Definition 12
3.2 Why efficient data structures are so important nowadays 12
3.3 Possible operations on data structures 13
3.4 Most common data structures 14

4 Algorithms **20**
4.1 Definition 20
4.2 Analysis of algorithms 20
4.3 Efficiency 21
4.4 Algorithm's running time complexity function estimation 22
4.5 Comparing algorithms 27
4.6 Representing complexity function with Big Oh for upper bounds 28
4.6.1 Definition of Big Oh 28
4.6.2 Role of big Oh in algorithms analysis 28
4.6.3 Best-case,average-case and worst-case input 28
4.6.4 Simplifying Big Oh 29
4.6.5 Converting complexity function to Big Oh notation 30
4.6.6 Comparing Algorithms according to their Big Oh notation 30
4.6.7 Final thought 32
4.7 Analyzing searching Algorithms:Linear search vs Binary search 32

5 Practical Part : Binary search Tree visualization **37**
5.1 Binary search Tree representation 37
5.2 Operations on binary search tree 38
5.3 Web page:Graphical representation of binary search tree 40

6 Survey Results **42**

7 Conclusion **44**
7.1 References 44

Appendices **46**

List of Figures

3.1 Prediction of exponential data growth toward 2020 and beyond 12
3.2 Worldwide internet user penetration from 2014 to 2021 13
3.3 One dimensional array 15
3.4 Two dimensional array 15
3.5 Stack illustration 16
3.6 Queue illustration 17
3.7 Modeling distances between the cities by graph 17
3.8 Binary search tree representation 18
3.9 Files and subdirectories in a directory representation using a tree 19
3.10 Tree can be used to represent hierarchy of the website 19

4.1 Running times of four algorithms for maximum subsequent sum in seconds 21
4.2 Program run in netbeans IDE .Messages printed 10 times,hence 10 steps 22
4.3 Graph showing the growth of functions as the value of input increases . . 32
4.4 Illustration of binary search, source!!!!(java book) 35
4.5 Binary search java code implemented in netbeans IDE 36

5.1 A binary search tree can be represented using a set of linked node 37
5.2 Two elements being inserted into the tree 40
5.3 Left panel containing control buttons and history area 41
5.4 Right panel that displays binary search tree 41

6.1 Respondents view on question regarding importance of focusing on efficiency in software design 42
6.2 Statics about to whom the respondents make software for 43
6.3 Statistics showing the domain of software the respondents were involved in 43

List of Tables

4.1 Algorithms comparisons based on the size of input 27
4.2 Multiplicative constant has no effect on growth rate 29
4.3 How functions change as the value of input increases 31
4.4 Binary search program run in java 36

Chapter 1

Introduction

The motivation to write a thesis about data structures and efficient algorithms design in java came after observing how applications are having to deal with large number of users and after observing how the amount of data to be processed is increasing at a high rate.This left me wondering whether some thing could be done in terms of storing data in a way that allows easy and quick access.For this reason,different methods of data storage and it's representation are discussed in this thesis.For data manipulations, different algorithms are analyzed using time complexity functions to determine the one which are more efficient than the others.Java programming language is used for implementation for this purpose.The website about one particular data storage is created where users can perform different tasks such as storing elements into the list,retrieving elements from the list and so on.At the end,the research is conducted in form of survey targeting the community involved in software design.The objective is to get the views on design focused on efficiency and how important this could be in tackling the issue of high number of users and huge amount of data.

Chapter 2

Thesis objectives and methodology

The main objective of this thesis is to analyze algorithms performance.

Throughout this thesis ,I will be demonstrating How algorithm design can affect per-performance .we will start by estimating algorithms efficiency mathematically, then different tasks in computer science will be analyzed such as searching and the comparison will be provided based on their common growth functions. any programming language may be used , here i will be using java programming language to implements algorithms(The details of java programming language are out of the scope of this thesis).The results will still be the same since the running time and memory taken by a program depend on how the algorithm is designed rather than the language used.That being said , anyone will be able to apply same methods used in this thesis using any other programming language such as C sharp or C++.

A simple website will be created at the end , for user to visually see and understand the benefits of how having the data stored in a certain manner can facilitate the manipulations on them.

The whole thesis will be written in a well known document preparation system called latex(The details about latex are out of the scope of this thesis)

At the end, a survey will be conducted. The questionnaire will contains questions designed to get the views from people involved in different areas of software development mostly on how the efficiency focused design can help deal with the large number of applications users and large amount of data storage.

Chapter 3

Data structures

3.1 Definition

The term data structures refer to a specialized way of organizing and storing data so that it can be accessed and manipulated efficiently.[2]

To understand what data structures are in real life, we can take an example of a program that sores information about the population of a country and frequently searches for a citizen using an identification number.The information should be organized in a particular way to allow fast search and this becomes even more important for countries with high population such as China with 1.379 billion inhabitants.

The efficient way to accomplish this task would be to use a map. A map is a type of data structure which is often considered as a container that stores elements along with their corresponding keys .For instance to search any particular citizen, the program would need to be provided with an identification number(key) to return the corresponding citizen.

3.2 Why efficient data structures are so important nowadays

Increase in data volume

More data has been created in past few years and the speed and data only keeps growing than ever before

Figure 3.1: Prediction of exponential data growth toward 2020 and beyond

Source: Insidebigdata [6]

As data grows, operations on data such as searching for an element in the list will become slower hence the need for the data to be well organized.

increase in application users

The number of users of applications has increased due to advancement of technology and devices.For instance 46.8 percent of global population has accessed internet in 2017 and this figure is expected to grow to 53.7 in 2021 according to world wide internet user penetration statistics.

An increase in internet user penetration means that web servers will have to deal with multiple requests at the same time .If the data is not well organized enough , this might lead to the web server failure.

To address those issues such as increase increase in volume of data,multiple user requests to name a few,data structures come to the rescue. It is possible to organize data in such way that in order for an application to search the data it would not have to search all the items.

There are many ways to organize data effectively depending on the operations that we want to perform. Later on in this thesis, we will see more in details some of the most frequent ones.

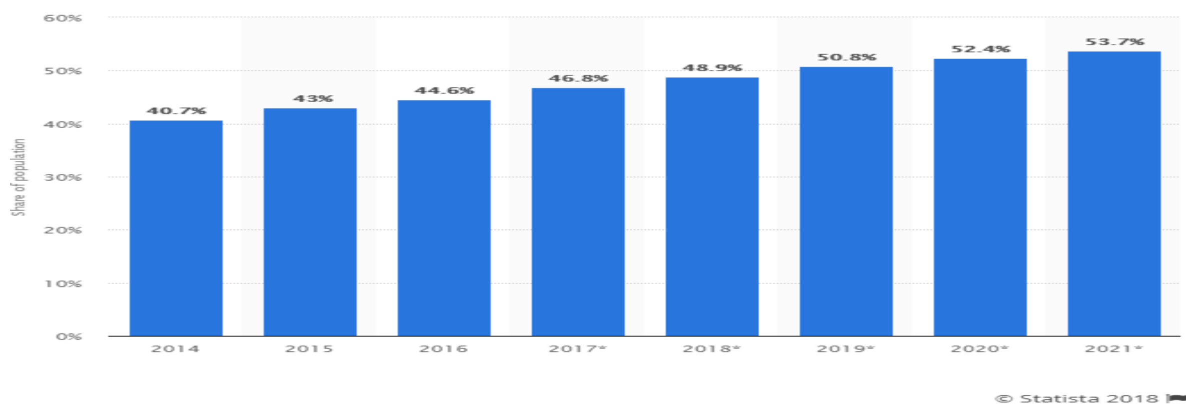

Figure 3.2: Worldwide internet user penetration from 2014 to 2021

Source: According to statista [8].

3.3 Possible operations on data structures

There are many different operations that can be performed on data structured depending on the tasks we want to accomplish, however some are common for most data structures.

Insert

We need to be able to add elements to the data collection.By performing insert operation, element can be added at some position in the list for example.Insert operation is common for many data structures.

Delete

The delete operation is basically the opposite of insert operation. we need to be able to remove some element at a given position from the storage and the delete operation does that.

Search

An other common operation to most data collection is the search operation. What search operation does is that it allow to find the location of the data item if it exists in the collection of data.Let's suppose a situation where we have the database of a country population that sores the citizens names with their social security number. By performing search operation, we can find the citizen corresponding to a given social security number.

Sort

Sorting in data structures basically refer to arrangement of data items in some logical order.for instance in case of numerical data such us the scores obtained by students in a given course can be arranged in ascending or descending order, on the other hand alphanumeric data can be arranged in dictionary order.We will see later how having data sorted facilitate other operations such as searching to be performed efficiently.

Merge

The last but not the least on my list of operations on data structures is merge. By performing merge operation,sorted data from two different files can be combined to make one file with sorted data.

3.4 Most common data structures

Data can be organized in different fashion depending on which task we want to accomplish.This section explain some of the most popular data structures in details.

Array

In computer programming, array refers to the collection of data of same type.It is one of the simplest data structures.

The elements in an array are accessed though their position also called index.Usually index uses a consecutive range of integers and this range always starts from 0 for most programming languages including java.

There exist one dimensional and multidimensional array.

In one dimensional array, elements are arranged in one row and can be accessed by a single index.

Multidimensional arrays are indexed by a umber of integers depending on how many rows and columns that make the array. For instance a 2 dimensional arrays are doubly indexed.

The most common types of arrays are one dimensional and two dimensional.

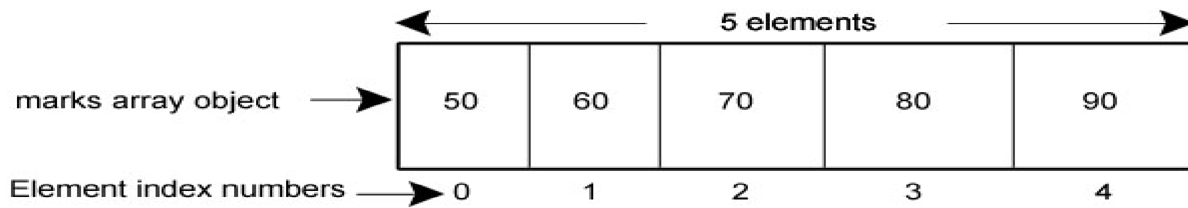

Figure 3.3: One dimensional array

Source: Internet

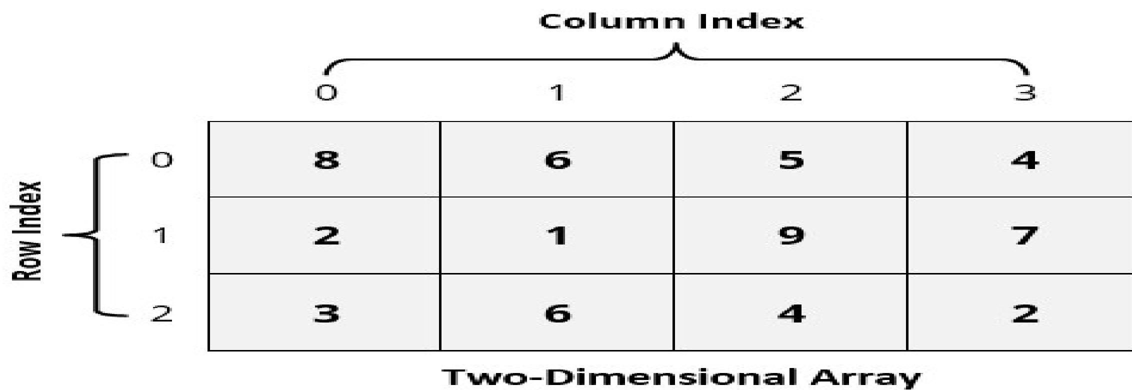

Figure 3.4: Two dimensional array

Source: Internet

List

A list is a popular data structure for storing data in a sequential order.
Real life examples of list:

- A list of students registered for a particular subject at university
- A list of rooms available for rent in a hotel
- A list of books a available in library, etc

All operations common for data structures that we have already seen can be performed on list as well, but the following operations are typical for most list

- Retrieving an element from a a list of elements
- Inserting a new element to the list
- Deleting an element from the list
- Find out whether a list contains elements
- Find the number of elements are in the list

- Find whether an element is in the list

  There are two ways to implement a list. One way is to use array to store elements and the other way is to use linked data structures that consists of nodes. The collection of nodes make a list.

Stack

The next type of data structure is stack. A stack is a linear structure in which operations such as accessing elements, deleting , inserting may be done at one end called the top. Stack structures are also called LIFO(last-in first-out) because elements are removed in stack in reverse order in which they were inserted to the stack.

A real life example is stacks is for example a pile of plates in the restaurant. An other example is a garage that is only one car wide.In order to remove one car from the garage, we would have to remove all the cars that came after it.

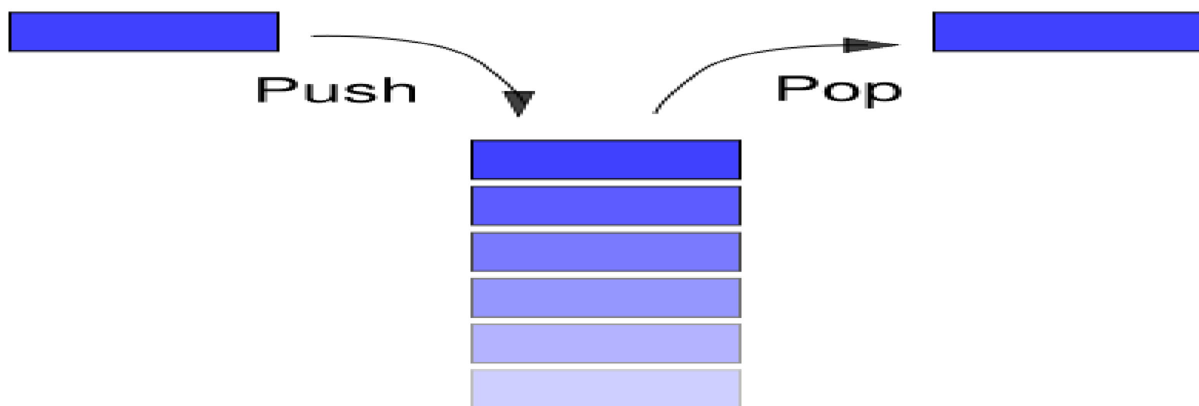

Figure 3.5: Stack illustration

Source: Internet

Pop is the technical term used for removing an element from the stack and push is used for adding element to the stack.

Queue

Like stack, queue is an other type of linear structure. Unlike for stack, insertion and deletion happens at different ends in case of a queue. Insertion takes place at one end called front and deletion takes place at an other end called rear.

Queues structures are also called FIFO(first-in first-out) because the first element to enter will be the first to leave the list. An often given analogy of the queue is the the counter in supermarket. The first customer to arrive on the queues is always the first to pay and the last to arrive will be the last to pay.

Enqueue means adding element to the front end of the queue and dequeue means removing element from the rear part of the queue.

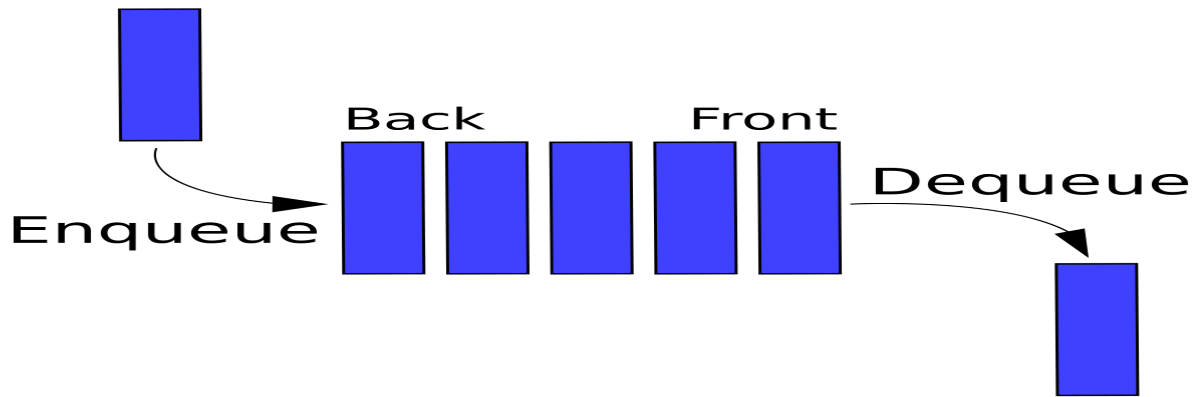

Figure 3.6: Queue illustration

Source: Internet

Graph

Graph is an other type of data structure that plays an important role in modeling real world problems.It consists of vertices called nodes and a set of ordered pair called edges.

Graphs are widely used in real life applications such as social media application. For example on facebook individuals are represented by nodes where each node contains information like user id, user name, user gender and information about the locale. Graphs are also used in networks , telephone network or circuit network.

Real world problems can be modeled by graphs, for example the problem of finding distances between the cities in a country can be modeled using graphs where the vertices(nodes) represent the cities and the edges represent the roads or distances between two adjacent cities. If we were further interested in finding out the closest cities, the problem can be reduced to finding the shortest path between tow vertices in a graph.

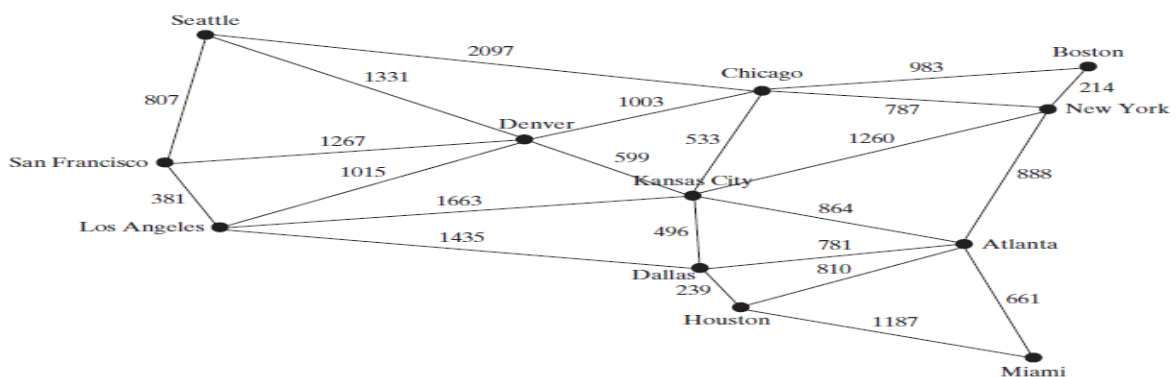

Figure 3.7: Modeling distances between the cities by graph

Source: [3]

Tree

Last but not the least in the list of data structures is Tree. A tree is a is a linked data structure where element are linked together through nodes.It provides a lot of applications due to its ability to arrange data in an hierarchical order .

In this chapter , we will be referring to one particular type of tree called binary tree where each node is characterized by its left reference , its right reference and its data element.The element on the top of a binary tree is called the root of the tree.

Later on we will talk about binary search tree, which is a special type of binary tree that fulfills the following conditions:

- Each node contains one key know as data and those keys can not be duplicated
- The key in the left subtree is smaller than the parent key
- The key in the right subtree is greater than the parent key.

For example a set of numbers 90,150,50,20,75,95,170 can be organized in binary search tree as follow

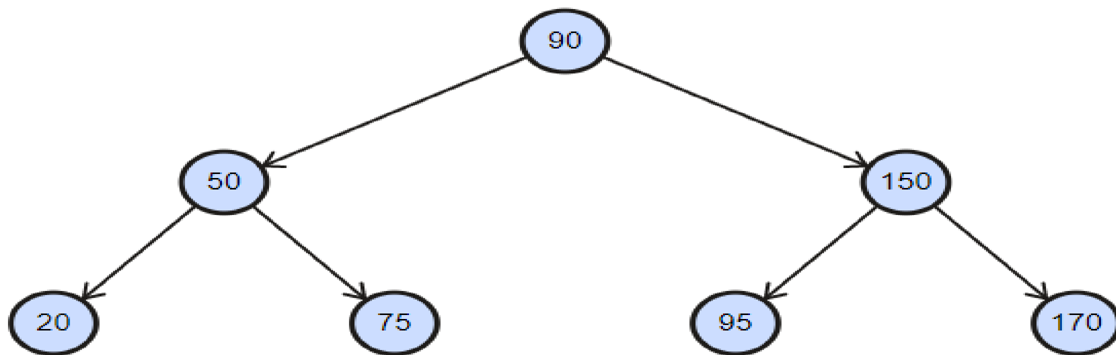

Figure 3.8: Binary search tree representation

Source: Author

We will see later on how binary search tree can be implemented . We will also see the advantages it has over other form of data storage in terms of retrieving data and so on.

*A*dvantages of tree structures

There are many applications of trees and below are the most popular ones

- Due to its structural relationships, trees are widely used in most operating systems to represent the directory structures and files in subdirectories

Also trees can be used to represent hierarchies such as the one found on website components or organizational structure of a company, etc

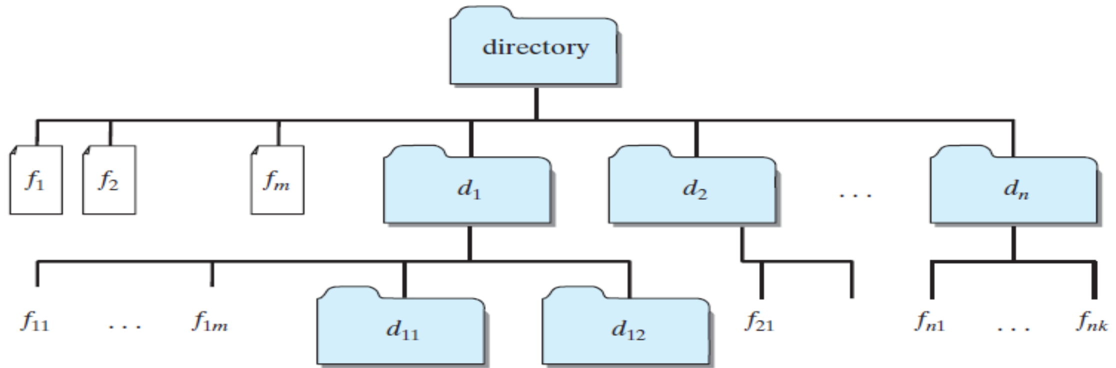

Figure 3.9: Files and subdirectories in a directory representation using a tree

Source: [3]

Figure 3.10: Tree can be used to represent hierarchy of the website

Source: Internet

Chapter 4

Algorithms

4.1 Definition

Simply put,algorithm is a clearly specified set of simple instructions to be followed to solve a problem in a finite period of time, especially by a computer [3].Historically used as tool for mathematical computation,algorithms are deeply connected with computer science and with data structures in particular.

A well designed algorithm is characterized by the following qualities:Receives zero or more inputs,produces atleast one output,consists of clear and unambiguous instructions,terminates after a finite number of instructions are executed.

4.2 Analysis of algorithms

The most important resource to analyze is generally the running time.Several factors such us the compiler, the computer used,algorithm used and the size of input affect the running time of a program,the first two(the compiler and computer used) are beyond any theoretical model so we will focus on the size of input and the type of algorithm. There may be more than one algorithm to accomplish one task,the objective here is to compare different algorithms in order to identify the one that takes shorter time to terminates and those that require few gigabytes of memory usage.An algorithm that solves a problem but require years to terminate is hardly of any use , likewise an algorithm that requires hundreds of gigabytes of main memory is not useful in most machines.It is especially important to be mindful of the algorithms and data-structures we use for applications that will process lots of data such as applications used for big data and internet of things. Typically, the size of the input is the main consideration.We define two functions $T_{avg}(n)$ $T_{worst}(n)$ as the average and worse-case running time respectively used by an algorithm on input of size n where average time is less than or equal to the time in worse-case scenario.

Occasionally,the best-case performance of an algorithm is Analyzed.However this is often of little interest because it does not represent typical behavior while worst-case represents a guarantee of performance on any possible input.

Generally,the quantity required is the worst-case time unless specified otherwise.One reason for this is that it provides a bound for all input including particularly bad input, which an average-case analysis does not provide.

As an example, lets consider a maximum subsequent sum problem. very interesting problem in computer science because there are so many algorithms to solve it and the performance of these algorithms varies drastically.In a maximum subsequent sum prob-

lem, were are given a one dimensional array that may contain both positive and negative integers and the task is to find the sum of contiguous subarray of numbers which has the largest sum.

Given(possibly negative)integers A_1,A_1,...,A_n, find the maximum value of $\sum_{k=i}^{j} A_k$ (for convenience,the maximum subsequent sum is 0 if all integers are negative)

For example, for input -2,11,-4,13,-5,-2, the answer is 20($A_2 through A_4$).

Several algorithms can be used to solve this kind of problem,the figure1 depicts the running time of the program using four different approaches.

	Algorithm Time			
Input Size	1 $O(N^3)$	2 $O(N^2)$	3 $O(N \log N)$	4 $O(N)$
N = 100	0.000159	0.000006	0.000005	0.000002
N = 1,000	0.095857	0.000371	0.000060	0.000022
N = 10,000	86.67	0.033322	0.000619	0.000222
N = 100,000	NA	3.33	0.006700	0.002205
N = 1,000,000	NA	NA	0.074870	0.022711

Figure 4.1: Running times of four algorithms for maximum subsequent sum in seconds

Source: [2]

As we can see from the program run, all algorithms seem to run in a very short period of time and if we were dealing with small amount of data , it really would not matter which algorithm we chose.But what would happen if we were dealing with big amount of data,let's say if we were interested to know the maximum subsequent sum for an array with more than 100000 elements? this is where clever algorithm design comes into play.

From the program run picture, the cubic algorithm get exhausted for the input of size n=100000 whereas for the same size of input,the linear algorithm produces output in a fraction of seconds,so if we were dealing with applications where the size of input required is more than 100000,clearly a linear algorithm design would the best choice of the four algorithms.

In the past years , applications were designed to process small amount of data compared to today's applications .we are living in time where the data to be processed by the systems is bigger than ever before and it only keeps increasing ,The IT systems need to be well designed in order to cope with this increase. At the same time,old systems needs to be optimized with sophisticated algorithms capable of processing big amount of data.

4.3 Efficiency

Measuring algorithm efficiency will help us get an idea on how well one algorithm perform against an other of similar nature. Some algorithms perform better than others.As we will see in more details, when searching an element from an array of elements ,binary search method which search for an element by comparing it with the middle element in the list,is

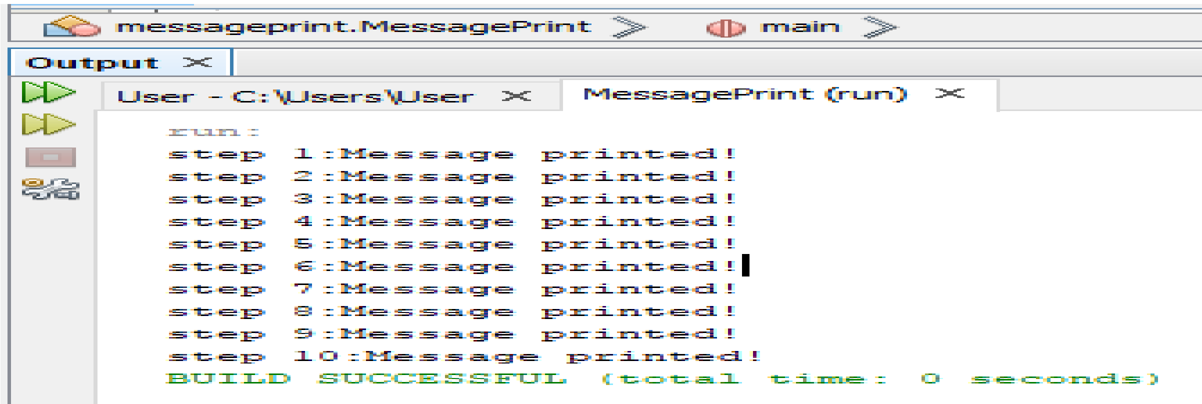

Figure 4.2: Program run in netbeans IDE .Messages printed 10 times,hence 10 steps

Source: Author

almost always more efficient than linear search method which has to go thorough all the elements one by one starting from the first element untill the key matching the element is found or an array is exhausted.

Algorithm efficiency is measured in terms of its time complexity by using complexity functions to abstract implementation and run time environment details.A complexity function shows the variance in algorithms time requirement for different input size(n).

In the following section, we are going to estimate the time required for a program and how that time can be reduced from days or years to fractions of a second.

[here to include a psedocode]

4.4 Algorithm's running time complexity function estimation

Lets begin by an example of a program that prints out a message on the screen a given number of times.The the java program to accomplish task can be expressed as below :

```
int i, limit;
for(i=1;i<=limit;i++){
System.out.print("message");
}
```

We can express the time complexity of an algorithm to accomplish the above task by specifying the time complexity function $t(n) = an + b$ where a is a constant multiplier representing the amount of time required to complete one loop iteration and b the setup time or the period required to prepare the machine for executing the program.In this example the time complexity function is clearly linear(for more about linear functions,refer to appendix on mathematics section).Since we want to abstract the machine details, we will often express the time complexity function in terms of the number of steps taken to complete.

How we define step can vary from one algorithm to an other so it is important to take care when defining algorithm's steps, so that the definition is meaningful and correlates with the algorithm's input size.In our printing message example,if we identified a single print instruction as the program's step,we would rewrite the time complexity function in terms of the printing step as follow: $t(n) = n$ which can be interpreted as for n times ,n steps are needed to print the message on the screen

Also for some programs such us sorting ,we can have steps for comparing and steps for exchanging.The complexity function will be based on the most dominant steps. It is easy to formulate a complexity function for a message printing program but it can become tricky and almost impossible for more complicated algorithms ,for this reason a set of rules has been devised.

Mathematical review

Before we see these rules , it is important that we take a look at some mathematical definitions

Function

In mathematics, a function is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output.An example is a function that relates a real number to it's cube.

linear function

Linear function is a function whose graph is a line in the plane.The main characteristic of linear function is that input is proportional to output which means that when the value of input variable is changed ,the change in the output is the constant multiple of the change in the input variable.

Example

$T(n) = an + b$

Quadratic function

In algebra, a quadratic function or simply a quadratic is a function with one or more variables where the highest degree is two.

Example

$T(n) = n^2 - 25$

Cubic function

A cubic function is a polynomial function where the highest degree is three

Example

$T(n) = n^3 - 9$

There are many more types of functions ,for now we will stop here and continue with the rules.Other functions types will be reviewed in later chapters.

Now that we know a bit about mathematical functions, let's see the rules to consider when analyzing the running time of different programs with loops, statements,conditions and more.

Rule 1:Loops

Generally, the running times of loops such as for loop is the running time of all statements inside the loop times the number of iterations. For a single loops algorithms in which the statements are executed a given number of times ,the complexity function is linear

Example

Consider the following program that prints the sum of numbers(whole positive integers) between 0 and n

```
int sum=0;//sum is zero at start
int n;
for(int i=0;i<=n;i++){
sum=sum+i;
}
```

From the above program , it is clear that the statement sum=sum+i; is executed in a constant time period that we can denote by the letter c.

If we consider the whole loop, the statement inside the curly brackets will be executed

n times since there are n iterations.Combining these two ideas,we are able to deduce the complexity function for the above program for printing the sum of numbers in any given range

$T(n) = (aconstantc) * n$

Where c represents the constant time required to execute the statement inside the loop and where n represents the number of iterations or the number of times the statement inside the loop is executed.

From the above single loop example example , the statements is executed only n times since there are n iterations, in other words the number of executions is proportional to the number of iterations hence the complexity function has the linear characteristic, However as we are going to see in the following examples,the same is not the case when the loops are nested inside other loops.

Double-nested loops

As we did with single loop , we will use an example to better understand how to find the complexity function for the double-nested loop.

Let's consider a situation where we want to find the sum of all the elements in two dimensional array. The following program does the task very well

```
int [][] array=new int[rows][cols];
int sum=0;

for(int i=0;i<rows;i++){

for(int j=0;j<cols;j++){
 sum=sum+array[i][j];
}
}
```

As for the single loop example ,it takes a constant time c for executing the statement sum=sum+array[i][j]; inside the nested loop. To keep things simple , we will consider the square matrix array where the number of rows is equal to the number of columns equal to n

The general form of such square matrix array would be

$$\begin{matrix} a_{00} & a_{01} & ... & a_{0n} \\ a_{10} & a_{11} & ... & a_{1n} \\ ... & ... & ... & ... \\ a_{n0} & a_{n1} & ... & a_{nn} \end{matrix}$$

By replacing cols and rows by n, our program becomes

```
int [][] array=new int[n][n];
int sum=0;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

 sum=sum+array[i][j];
```

```
}

}
```

We are ready now to formulate the complexity function for our problem. It takes a constant c to execute the statement inside the inner loop and since the inner loop executes n times the statement has to be executed for each iteration.

At this point,the complexity function is the same as for a single loop
$T(n) = (aconstantc) * n$

But how about the outer loop? the outer loop executes n times which means that for every iteration , the whole inner loop is executed n times.This brings us to the final complexity function for a nested loop

$T(n) = (aconstantc) * n * n$ or $T(n) = (aconstantc) * n^2$

This function is quadratic.If we double the size of input, the time will be quadrupled .The quadratic function grows faster than linear functions in general.All nested loops more or less behave in this way.

By following the same analysis, it can be proven that the complexity function for triple-nested loops is cubic.The pattern continues for even higher nested loops.

Rule 2:Consecutive statements

In the previous sections , we analyzed the complexity functions for loops(single loops and nested loops), in this section we are going to see how to determine the complexity function for consecutive statements.

Consider the following program:

```
//first loop
for (j = 1; j <= 10; j++)

{

k = k + 4;

}

//second loop

for (i = 1; i <= n; i++)
{

for (j = 1; j <= 20; j++)
{

k = k + i + j;

}

}
```

How would we make the complexity function for the above program? to answer this question, we need to analyze the program code by code .As we can see, the program is composed of one single loop and one nested loop.

For the single loop part

We have seen that it takes a constant time c to execute a single statement, This the same for the statement k=k+4; inside the single loop. Since the loop has 10 iterations , the execution time is 10 times a constant c.

For double loop

It takes a constant time c to execute the statement k = k + i + j;.There are 20 iterations in inner loop , therefore the execution time for the statement k = k + i + j; becomes a constant 20 times. The outer loop has n iterations , which means that inner loop will be executed n times therefore the execution time for the statement k = k + i + j; becomes a constant times 20 times n(where n is the number of iterations)

Now that we have analyzed the complexity function for the fragments of the program,we can find the complexity function of the whole programs by just adding the functions together.

Hence
$T(n) = 10 * c + 20 * c * n$

Rule 3:Selection statement:If/Else

Now its time to analyze the running time of some condition statements Consider the following program [2]:

```
if (list.contains(e))

{

System.out.println(e);

}

else
for (Object t: list)

 {

System.out.println(t);

}
```

The program does a simple thing ,it checks whether an element is in the list , if the element is found ,the program prints it out. If the element is not found, the program prints everything that is found in the list.

Assuming that the list has n elements, the test is executed n times(assuming the worse case scenario).The loop in the else clause takes n time , So the time complexity for the entire statement can be determined as if test time plus worse case time (if clause/else clause)

$T(n) = constant * n + constant * n$

Usually when performing algorithm analysis, we are interested in algorithm's efficiency as n tends to a very larger value. Later on in this thesis,we will see how all the complexity functions can be estimated for larger size of input using the method called Big Oh notation

4.5 Comparing algorithms

Now that we have seen complexity functions for different algorithms , it's time to see how we can compare two or more algorithms in order to choose the best one for solving a particular task.

It is particularly important to ensure a fair comparison when comparing algorithms. It would not make sense to compare a sorting algorithm with a searching algorithm , therefore algorithms of similar nature should be compared.

For example it would logically make sense to compare two algorithms that search for an element from a list of elements(Binary search and linear search,for instance). An other example is to compare two or more algorithms that sorts elements of a list in any preferred order.

An other factor that can influence the comparison result is the size of input.This is mostly due to the fact that complexity functions are not linear which means that one algorithm can be the best for small size of input but worst for a bigger size of input.

Let's consider a situation where the time complexities for two algorithms that solve a particular problem are given by the functions below

$T_1(n) = 2n^2 + 3n$

$T_2(n) = 30n + 1$

The best way to see the performance of the two functions is to draw a table and see what will be the output for different values of n.

n	$T_1(n)$	$T_1(n)$
1	5	31
2	14	61
3	27	91
4	44	121
5	65	151
6	90	181
7	117	211
8	152	241
9	189	271
10	230	301
11	275	331
12	324	361
13	377	391
14	432	421
15	495	450

Table 4.1: Algorithms comparisons based on the size of input

Source: Author

As we can see from the data in the table above,algorithm with complexity function $T_1(n)$ has smaller output than the second function $T_2(n)$ for input size n less than 14.For input size n from 14 and above however, we see that the output of function $T_1(n)$ becomes bigger than that of $T_2(n)$.

The difference becomes even more obvious for larger input since quadratic grows faster in general than linear function.

If we were interested in problems where the size of n was small,Algorithm with function $T_1(n)$ would be a better choice

4.6 Representing complexity function with Big Oh for upper bounds

Computer scientists use asymptotic analysis to compare algorithms in situation where n is big enough or tends to infinity. The notation called Big Oh is used to represent the complexity functions when n tends to infinity.

4.6.1 Definition of Big Oh

A function $f(n)$ is $O(g(n))$ if and only if there exist two constants c and n_o such that

$f(n) <= cg(n)$ for all $n >= n_o$

All $f(n), g(n), n, c, n_o$ must be positive

Where

$f(n)$ represents algorithm's running time (as estimated in previous sections) $O(g(n))$ represents Big Oh representation for $f(n)$.

4.6.2 Role of big Oh in algorithms analysis

The big Oh notation is used to represent the order of magnitude of different algorithms according to their growth rates . Consider a linear search for example. A linear search is an algorithm that compares the key with elements in the array of n elements sequentially until the match is found or the end is reached without any match. If the match is found ,the number of comparisons will depend on the position of the element in the array but on average we can say that $n/2$ comparisons will be needed . If there is no match found however , the number of comparisons will definitely be equal to n since the algorithm will have to compare the key with all elements from the beginning till the end.In either case , we would expect the number of comparisons to double if the size of array were doubled,hence the algorithm's linearity behavior.The growth rate has an order of magnitude n. The order of magnitude for linear search algorithm can be represented by Big Oh notation as $O(n)$ which is pronounced as "order of n".

4.6.3 Best-case,average-case and worst-case input

The running time of an algorithms such us linear such algorithm depends on the input.Input that requires the smallest number of comparisons will make execution time shorter and can be said to be the best-case input. On the other hand, an input that requires the largest number of comparisons will make execution time longer and can be said to be the worse-case input.

Average-case analysis tries to determine average execution time among all possible inputs.

Worse-case input is very useful because it helps us to get an idea about execution time of an algorithm in worse case scenario and it assures that no other input will cause the algorithm to execute in bigger time.

4.6.4 Simplifying Big Oh

When using Big Oh notation , we are always interested in growth rate of complexity functions . It is therefore no surprise that the Big Oh representation for linear search algorithm in average-case and worst-case is the same$O(n)$ time since both functions $n/2$ and n grow at the same rate (Table below).

n	$n/2$	n
100	50	100
200	100	200
Growth rate	2	2

Table 4.2: Multiplicative constant has no effect on growth rate

Source: Author

1. *Omitting multiplicative constants*

   Because Big Oh represents the growth rate of complexity function and any multiplicative constant has no effect on growth rate of any function,the constant will always be ignored.

   From the table above, the growth rate for $n/2$ is the same as the growth rate for n and is equal to 2.As we can see, the constant 1/2 has no influence on the growth rate.Therefore $O(n)$=$O(n/2)$.

2. *Ignoring non dominating part*
   Let's imagine a situation where we are interested in determining the maximum number in an array that contains n elements.The algorithm than accomplish this tasks assume that the first element is the maximum number and compares this number with all other elements in the array to verify if the element is indeed the maximum. If there are 2 elements in array,algorithm will need 1 comparisons. For 3 elements 2 comparisons will be needed. Generally,it will take n-1 comparisons to find the maximum number from an array of n elements. Algorithm analysis is for larger input and as n grows larger, it dominates to complexity. With Big Oh notation, the dominated part of the complexity function can be ignored.Therefore the Big Oh representation for finding the maximum number can be written as $O(n)$.

   Generally, it is common to ignore all terms except the largest and all any constant multiplier when representing any complexity function using Big Oh notation.

3. *Constant time algorithm*

   With Big Oh notation, we try to estimate the running time of an algorithm according to the size of an input. For some situations however, input and running time might not have any relation at all.

Consider an algorithm that prints an element at any position of an array of size n. The execution time will always be the same whether the size of an array is increased or reduced.In situations like this where the size of input is not related to the running time,an algorithm is said to take constant time and it is always represented as $O(1)$

4.6.5 Converting complexity function to Big Oh notation

Now that we have the notion of Big Oh , its time to go back to the section where we formulated the complexity functions for loops , statements and conditions. All those functions can be represented using Big Oh notation. Here the rule about highlighting part and ignoring multiplicative part of complexity functions play a big part.

1. Single loop
   After analysis, we found that the complexity function for algorithms involving single loop was of the form $T(n) = c * n$

   where n represented the number of iterations and c was the constant time take to execute the statement inside the loop. As we have already seen,Big oh notation allows us to throw away any multiplicative constant since it has no impact on the growth rate of the complexity function. Applying the same rule, the Big Oh for single loop can be written as $O(n)$.

   $c * n = O(n)$

2. Double loop

   We have seen that the complexity function a double loop can be formulated as $T(n) = c * n^2$ where n represents the number of iterations for both inner and outer loops , C a constant time required to execute the statement inside inner loop.

   As for any linear function,any multiplicative constant does not affect the growth rate of a quadratic function. For this reason, the constant c can be ignored and the Big Oh notation for the above complexity function can be written as $O(n^2)$.

3. Consecutive statement
   The complexity function determined after analyzing the algorithm was $T(n) = 10 * c + 20 * c * n$. By ignoring the non dominating part(10*c) and multiplicative constant(20*c), the Big Oh representation for the above function can be written as $O(n)$.

4. Condition if/else

   The complexity function $T(n) = constant * n + constant * n$ found after analyzing the example of an if/else condition can be represented in Big Oh notion as $O(n)$ after getting rid off all constants from the function

4.6.6 Comparing Algorithms according to their Big Oh notation

We have already already analyzed to complexity functions of several algorithms in previous sections .In practice when comparing two or more algorithms to solve a task, they will probably have different growth rate .As we will see later , the two most common algorithms used to search for an element from a list of elements grow at different rate.

To be able to chose the best algorithm, we need to have information about the growth rate of all candidates algorithms and then we can choose the one that grows at a slower rate.

During analysis,we found out that some algorithms complexity functions were constant,others linear and others were quadratic as found in case of doubly nested loop.Normally algorithms can be of any form . As we will see in the case of binary search algorithm,the complexity function is logarithmic. Other complexity might be exponential, others log-linear or cubic etc .

Knowing how each complexity function grows will help us to determine the one that grows at lower rate among alternatives.

The best way to find out the rate at which the functions grow as n gets larger, is to find the values of those functions for different values of input n, and observe how the values of the functions change. we will do this with the help of a table .

Big Oh function	Name of function	$n = 50$	$n = 100$	$T(100)/T(50)$
$O(1)$	Constant time	1	1	1
$O(\log n)$	Logarithmic	5.64	6.64	1.18
$O(n)$	Linear	50	100	2
$O(n \log n)$	Log-linear	282	664	2.35
$O(n^2)$	Quadratic	2500	10000	4
$O(n^3)$	Cubic	50^3	100^3	8
$O(2^n)$	Exponential	2^{50}	2^{100}	$1.13e + 15$

Table 4.3: How functions change as the value of input increases

Source: Author

Outcome analysis

From the results found in the rate columns of the table , we can see how the various functions grow as n increase from 50 to 100.

On the first row , we have constant time function. As we have already seen during Big Oh notation analysis,the time execution time is independent of the input size and this the reason why the value of the function stays the same both for $n = 50$ and $n = 100$

On the second row however,as the value of n doubles, the value of the function increases and we can see that the function grows at the rate of 1.8. Here we assumed the logarithmic base of two for both values of n.

The linear time complexity on the third row behaves as we would expect for any linear function. As the size of input doubles, the value of the time doubles as well. In other words the time of algorithm is said to be directly proportional to the size of input.

The fourth row of the table corresponds to an other type of complexity function called log-linear function. As we can see from the values in the growth rate table,this function grows faster than both linear and logarithmic functions.

The 5th row corresponds to the quadratic complexity. As for any quadratic functions , the value of the function is quadrupled when the value of input size is doubled. By doubling the value of the input size from 50 to 100, the value of the function is quadrupled(from 2500 to 10000). The growth rate for this function is 4 as shown in the growth rate column of the table.

In the 6th row of the table,we have the cubic function.Cubic functions grow faster than quadratic and all other previous functions as we can see from the table. By doubling the size of input n, the value of the time is increased by the factor of 2^3.

Finally the last row of the table shows how exponential complexity functions tend to grow as the value of input size gets bigger and bigger. By just doubling the input size from 50 to 100, the value of the function changes from 2^{50} to 2^{100} which remarkably a big increase.Exponential complexity functions grow at a higher rate in comparison to other functions in given in the table.

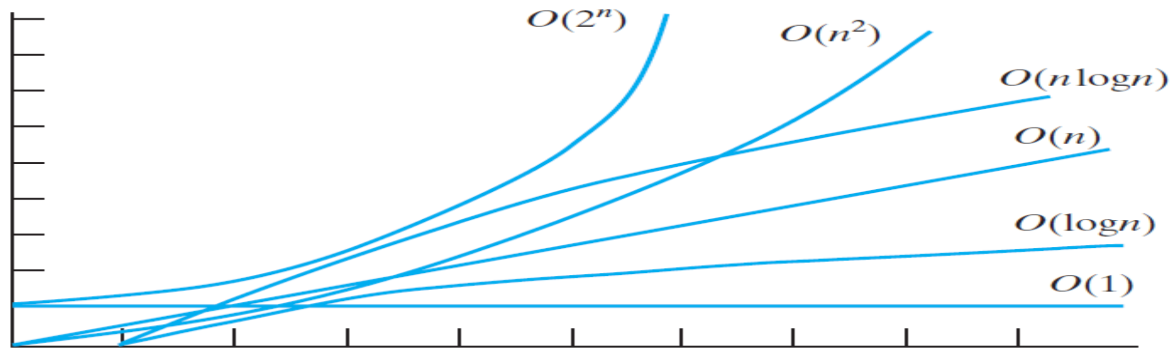

Figure 4.3: Graph showing the growth of functions as the value of input increases

Source: [3]

4.6.7 Final thought

Just because a function grows slower than other does not mean that this function is always the best choice in all situations, it also depends on the range of the size of the input value. For instance cubic function tends to grow faster than exponential functions for smaller value of input size n.

In most cases when analyzing algorithms efficiency, we are interested in algorithm's behavior for a very large input size of n, with that in consideration, algorithms with growth rate such as the one seen for exponential function will always be avoided

4.7 Analyzing searching Algorithms:Linear search vs Binary search

Searching data is a common task in most applications. Searching is basically the process of looking for a specific element from a a list of elements. There are many everyday life situations where we are interested to find an element from a long list of elements. Searching for a word in a dictionary for instance or searching for number from a long list of numbers etc.

The two algorithms most common to accomplish the task of searching are linear search and Binary search algorithms.Both linear and binary search algorithms accomplish the same thing but in different ways . They all scan the list of elements until the match is found.If the match is found the algorithms stop and they return the match and if no match found,that means the element searched is not in the list.In this section,we are going how these two techniques differ in terms of efficiency by analyzing their complexity functions for a lager size of input.

Linear Search

The linear search compares the key with each element in the list starting from the beginning.The comparison continues until the key matches the element in the list. If the element is not found ,the searches returns a negative value usually -1 to inform that the match was not found

Implementation in java

The linear search algorithm can easily be implemented in java programming language as follows

```
public class LinearSearch{

public static void main(){


}

public static int linSeach(int[] list , int key){


for(int i=0;i<list.length;i++){

if(key==list[i])
return i; //the match is found, position returned

else
    return -1 // No match found . -1 returned
}


}

}
```

To better understand output of this program,lets suppose the list array contains elements 44,55,22,77,10 and that we want to search for number 77 from the list. By passing the arguments list and key to the method linSearch,the number 3 will be returned and this corresponds to the position of the key element 77 in the list(the first number of array in java have index 0,the second element 1, and so on) . If we were looking for a number which is not in the list however, the linSearch method would return the number -1 meaning that element was not found.

Complexity function of linear search

Since linear search compares the key element with each element sequentially, the elements in array do not need to be sorted.The execution time increases linearly as the size of array increases. This approach of having to compare the key with each element in array becomes inefficient as the size of array becomes larger. For example if the element we were looking for was the last in the list(worse-case), the algorithm would have to make unnecessary comparisons before finding the match.In the next section, we will talk about an other approach called binary search and see some of the advantages it has over linear search approach.

Binary search

After discussing about linear search approach in previous section, its time to talk about an other searching approach called binary search.Binary search approach accomplishes the same task as binary search but just in different way.One difference is that for binary search to work , elements in the list must be ordered either in ascending or descending order.Let's see how this approach works in both cases

We are going to assume that elements of the list are arranged in ascending order.For the elements in descending order, binary search approach works in more or less the same way(comparing the key element with the middle element and eliminating half of the list if necessary).

When the elements of the list are ordered in ascending order,the binary search approach compares the key element with the element in the middle and from this comparison, three cases arise:

key element is less than middle element

After comparison if the key element is less than the middle element,binary search will continue to search for the key element only in the first half of the list and an the half will be eliminated from the search area.

The key element is equal to the middle element

After comparison, if the key element is equal to the middle element, the match is found and no further search is required.

The key element is greater than the middle element

The last scenario is when after comparison the key element is greater than the middle element. In this case, The binary search eliminates the first half of the list and continue to search in the second half.

It is clear that after each comparison,half of the list is eliminated. We are going to assume that there are n elements in the list.After the first comparison, $n/2$ elements are left for consideration for next comparison and after second comparison$n/4$elements are left for consideration for next comparison.After the kth comparisons,$n/2^k$ elements are left for consideration for the next comparison.When k is equal to log_2n, only one element in the list is left and just one more last comparison is needed. Therefore, in the worst case scenario,it would take $log_2n + 1$ comparisons to find an element from the list using the binary search approach on a sorted list of elements.For instance if we consider a long list of 1048576(2^20)elements,it would require 21 comparisons in worst case for binary search approach and 1048575 comparisons for linear search approach.

As a clear demonstration of how binary search works, let's consider an array of elements 2,4,7,10,11,45,50,59,60,66,69,70,79.the portion of the array being searched shrinks by half after each comparison.We need to keep track of the index of the first, last and middle element for each search portion.Let's use low,mid and high for index of element on the first,middle and last element respectively.Initially,low is 0 and high is list.length-1(index of the last element in the list) and mid is $(low + high)/2$

the figure below shows how to find the key 11 from the list using binary search approach.

Binary search implementation in java

```
public class BinarySearch{
```

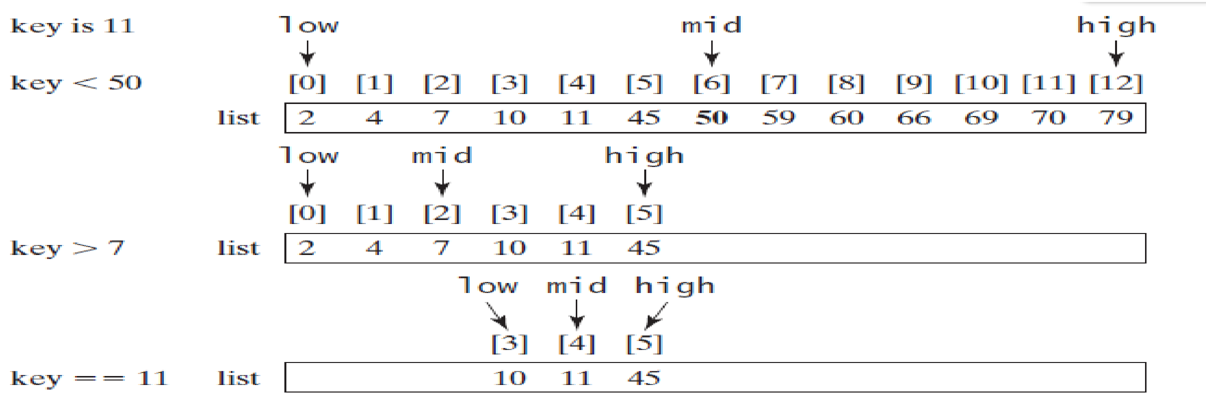

Figure 4.4: Illustration of binary search, source!!!!(java book)

Source: [3]

```
    public static void main(String[] args){

}
  Public static int binarySearch(int [] list ,int key){

  int low=0;
  int high=list.length-1;
   while(high >= low){
   int mid=(high+low)/2;
   if(key <= list[mid])
   high=mid-1;

   else if (key==list[mid])
   return mid;

   else
        low=mid+1;
   }
  return -low-1 //a negative number returned indicating that the key was no

  }

}
```

For better understanding of what is happening , we will call the binarySearch method with the list 2,4,8,16,32,64,128,256,512.The following table illustrates the values for low, high and the value returned which is the index of the key element.

As conclusion, both linear search and binary search can be useful depending on application.If the data we are interested in are already sorted,The binary sort approach would be a better choice since the search is done in few comparisons in comparison to the linear search.For some type of data structure such as linked list,the binary search approach is not applicable since there is no exact location of the middle element.On the other hand, if the data to be searched is not arranged in certain order,the linear search approach would be the best choice.Since the time complexity to sort elements in the list($O(logn)$) is usually greater than the time complexity required to search for an element, it is obvi-

```
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          int [] list={2,4,8,16,32,64,128,256,512};
20          System.out.println(binarySearch(list,2));
21      }
22      public static int  binarySearch(int[] list,int key){
23          int val;
24          int low=0;
25          int high=list.length-1;
26          while(high>=low){
27              int mid=(low+high)/2;
28              if(key<list[mid])
29                  high=mid-1;
30              else if(key==list[mid])
31                  return mid;
32              else
33                  low=mid+1;
34  
35          }
36          System.out.println("low value is "+low+" and high value is"+high);
37          return -low-1; //key not found
38      }
39  }
```

Figure 4.5: Binary search java code implemented in netbeans IDE

Source: Author

Key value	low value	high value	returned value
64	5	5	5
100	6	5	-7
512	8	8	8
2	0	0	0

Table 4.4: Binary search program run in java

Source: Author

ously more efficient to apply linear search approach to in-ordered list rather than having to sort the list and apply binary search approach afterward.

Chapter 5

Practical Part : Binary search Tree visualization

5.1 Binary search Tree representation

In the chapter about data structures , we talked about how data are arranged in tree structure. We also saw some of the advantages of arranging data into tree structures in comparison to other forms of data storage. This section is dedicated to one of the most important data structures called binary search tree(often shortened as BST) which is special type of binary tree with the following properties:

- Each node contains one key known as data and those keys can not be duplicated
- The key in the left subtree is smaller than the parent key
- The key in the right subtree is greater than the parent key.

A binary tree can be represented by linking nodes together.Each node is linked to the node on the left called left child and on the right to the node called right child.

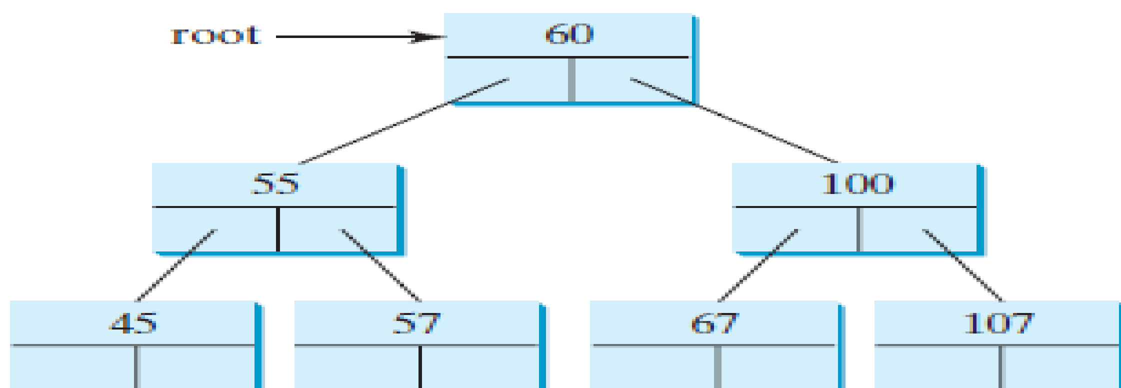

Figure 5.1: A binary search tree can be represented using a set of linked node

Source: [3]

How do we represent a node in java programming language?.We can define a node using a class as follows:

```
class TreeNode<E>{
```

```
E element;

TreeNode<E> left; //left represents left child
TreeNode<E> right; // right represents right child

public TreeNode(E e){ //constructor for a specified element

element=e;

}

}
```

The root variable as shown on the top of the tree(figure 5.1) is the root node of the tree.It is null in case there are no elements in the tree.

Now lest creates the nodes of the tree shown in our example in figure 5.1 using the TreeNode class in previous code.

```
//lets create the root node first

TreeNode<Integer> root=new TreeNode<integer>(new Integer(60));

//next we create the left child node

root.left=new TreeNode<Integer>(new Integer(55));
//Then we create the right child node
root.right=new TreeNode<Integer>(new Integer(100));
```

5.2 Operations on binary search tree

Searching of an element in the tree

Retrieving an element from the binary search tree is fast mainly due to the fact that elements are arranged in special order where elements in the value in the left child node is smaller than the value in the parent node and the value of the element in the right child node is greater that the value of the the value of the element in the parent node. Searching for an element in binary search tree structure is comparing the element we are looking for with every node value until the match is found.The search starts from the root of the tree and scan down until the match is found or an empty subtree is reached in which case there is no match.

In the section about binary search approach, we found out that the key element is compared with the middle element.If the key is equal to the middle element,the match is found, and we stop the search. If the key is less than the middle element, all the elements after the middle elements are ignored,and we only continue our search in the first half.How does searching in the binary search tree works?

Like we have seen for binary search on arrays, searching an element in binary search tree is also based on comparisons.Firs we compare the key element with the root element. If the match is found, we stop.If the key is less than the root element,we continue searching

in the left hand side of the root element otherwise we search in the right hand side of the element. The search goes on until the match is found.

Implementation of the above can be accomplished in java using the following code[3].

```
public boolean search(E element){
TreeNode<E> current=root; //we start from root

while(current !=null){

if(element < current.element){
  current=current.left;
  }
  else if(element>current.element){
  current=current.right;
  }
else
     return true; //there is a match
}
 return false;
 }
```

Adding an element into binary search tree

To be able to insert an element into a binary search tree structure , algorithm needs to locate the parent for the new node first.

The algorithm can be implemented in java using the following code ;

```
boolean insert(E e){

if(Tree is empty){
//e becomes the root node
}
else
{
//we locate the parent node
parent=current=root;

while(current != null){
if(e < the value in the current.element ){
parent =current; //keep the parent
current=current.left; go left
}
else if(e > the value in current.element){
 parent=current; keep the parent
 current=current.right; // go right
}
else return false; //duplicate inserted.

return true; //node inserted
}
```

}

}

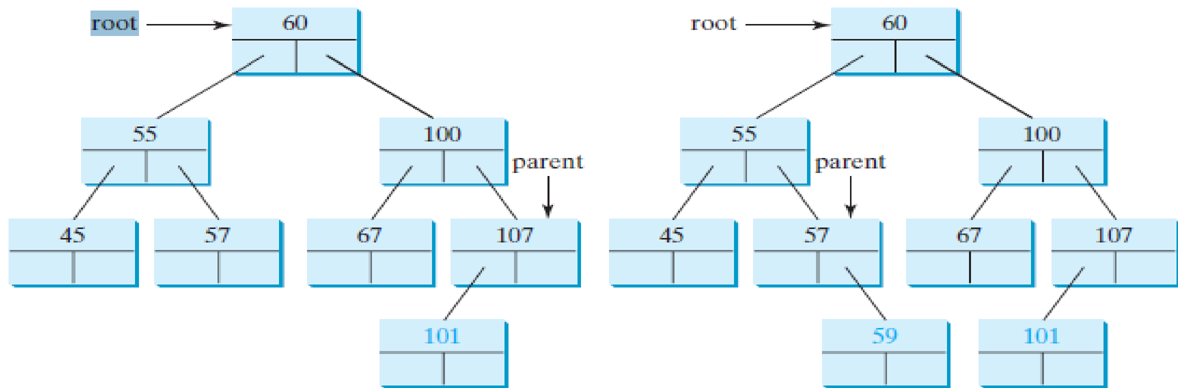

Figure 5.2: Two elements being inserted into the tree

Source: [3]

5.3 Web page:Graphical representation of binary search tree

As part of practical part of this bachelor thesis ,a simple web page has been created to show graphically how the binary search tree works.The operations on data structures already discussed such as inserting element, deleting element,retrieving an element etc can be performed on this web page.The website can be found on this url https://bstviz.000webhostapp.com/

The website is created with advanced technology that allows users to see the operations being performed graphically. The interface is made of two main parts:

The left part contains the control buttons allowing users to perform different tasks.Above the control buttons is a big area dedicated for showing the history of all operations performed

The right part contains a big area for showing the graphical representation of the binary search tree structure.By adding an element using the add element button in the left panel,users get to see element being added to the right panel with slow animation.

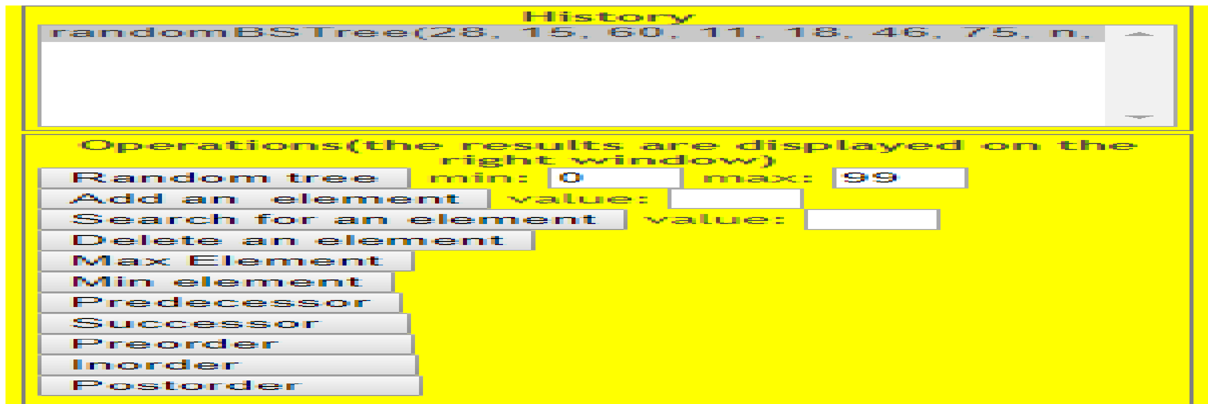

Figure 5.3: Left panel containing control buttons and history area

Source: Author

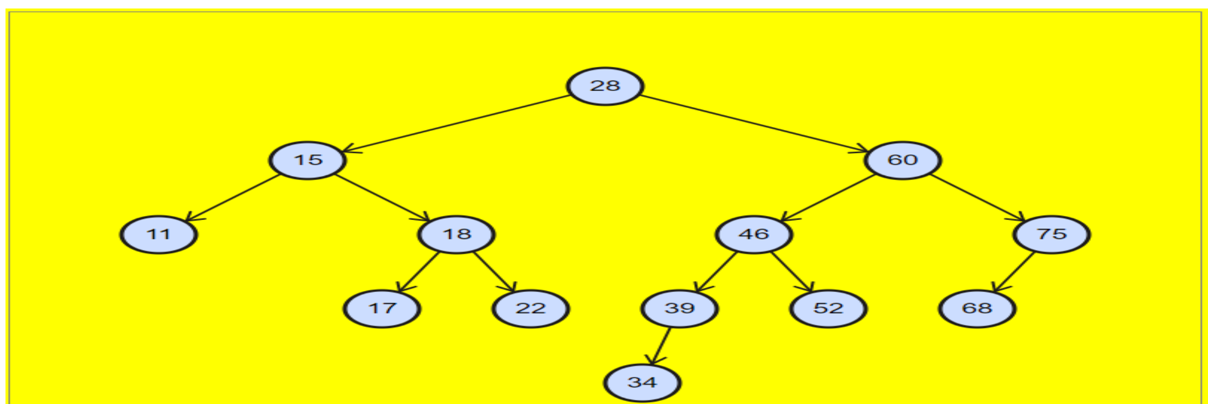

Figure 5.4: Right panel that displays binary search tree

Source: Author

Chapter 6

Survey Results

As part of this bachelor thesis , i conducted a survey. The aim of the survey was to investigate how often people involved in software development(programmers,developers,engineers,testers, etc)care about the performance of the software they design. The questionnaire were composed of different questions designed to get respondent's views on things such as efficiency, effectiveness and so on.The list of questions asked and feedback from respondents can be found in the appendix of this thesis.

One of the questions asked was about efficiency and effectiveness in software design .The objective of the question was to let the respondents decide which one is important and then give opinion about their choice.60% of the respondents went for efficiency,40% for effectiveness and the remaining 10% said that they did not consider neither efficiency nor effectiveness.

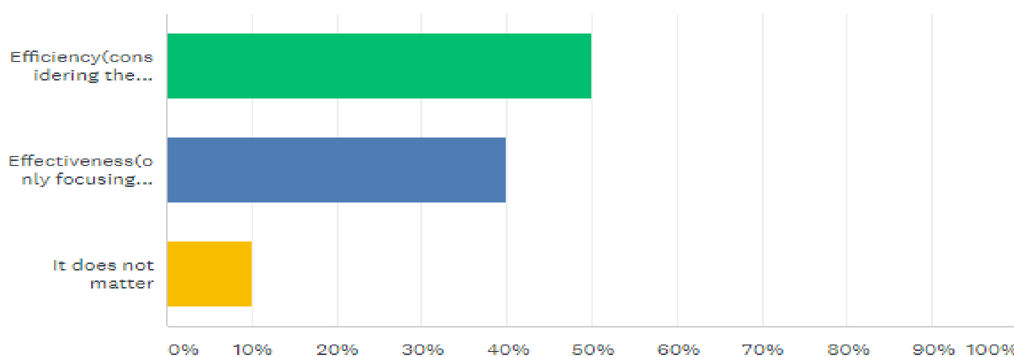

Figure 6.1: Respondents view on question regarding importance of focusing on efficiency in software design

Source: Author

One of the respondents who chose efficiency said that the reason why they focus on efficiency is that their products are used by a large number of people and that their applications usually have to support a big amount of data.On the other hand,one of the respondents who chose effectiveness said that the reason why is that their products are intended for a small group of users and that it would take unnecessary long time trying to analyze the efficiency.Finally those who said they did not care are the ones who just create small software for fun or not intended to the real users.

Looking back to the survey results and feedback from the respondents, it is clear that their choice dependent mostly on two things: Firstly the number of users expected to be

using their products and secondly the amount of data their applications are supposed to handle.The fact that efficiency came on top make complete sense since the majority of people who took the survey were involved designing software products for real end users.

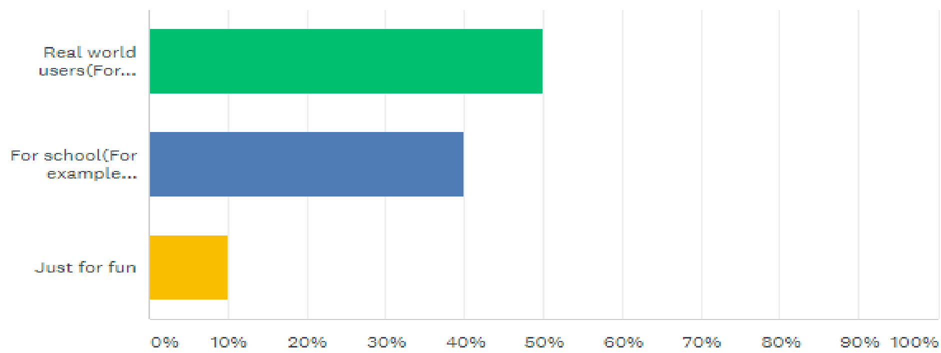

Figure 6.2: Statics about to whom the respondents make software for

Source: Author

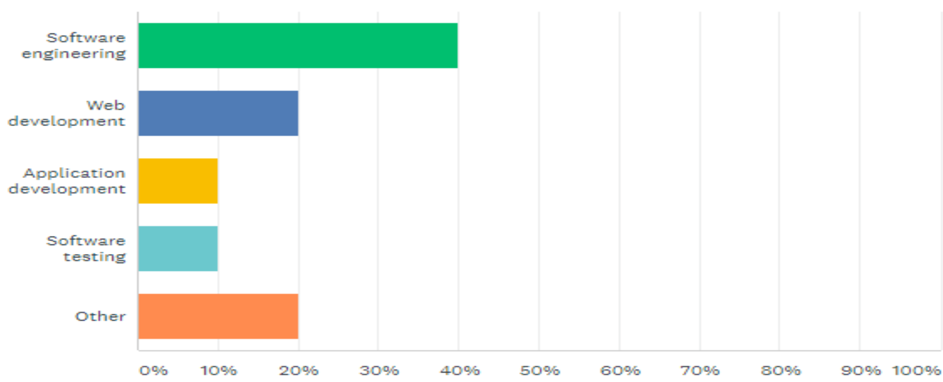

Figure 6.3: Statistics showing the domain of software the respondents were involved in

Source: Author

Chapter 7

Conclusion

In this thesis,we talked about different ways of storing data in memory and the analysis of different algorithms on data were provided.In the data structure chapter , we saw different ways in which data can be stored and some of the operations that can be performed on them.The algorithms chapter focused on different techniques on how to estimate the running time of different algorithms based on their growth functions.The results from the tables in conducted experiments showed us how important it is to consider the amount of data the application needs to work with before deciding which algorithm to use since one algorithm may be bad option for small data set but best for large data set.

The practical part of this thesis focused on one special data structure called binary search tree.The theory on how data are stored and it's implementation in java environment was given. The website was created with objective to demonstrate how data is stored in binary data structure and which operations can be performed on them.The website can be accessed here https://bstviz.000webhostapp.com/

Finally,the results from the research and the feedback from respondents showed how designing software with efficiency in mind can help deal with high increase in number of users and high volume of data that needs to be supported.The sample size of 10 responses was enough since the main goal of the survey was to get feedback from respondents regarding the role of efficient focused design on applications performance.

7.1 References

Bibliography

[1] 23. D. E. Knuth, The Art of Computer Programming: Vol. 3: Sorting and Searching, 2d ed., Addison-Wesley, Reading, Mass., 1998,ISBN-13: 9780201896855

[2] Data structures and algorithms analysis in java , by mark allen weiss, 3rd edition .Florida international univesity,ISBN-13: 9780132576277

[3] Introduction to java programming , 8th edition , by Y.Daniel Liang. Armstrong atlantic state university, ISBN-13: 978-0-13-213080-6

[4] Java ee 6 development within netbeans 7 by David R.heffelfinger, first published 2011, ISBN 978-1-849512-70-1

[5] Creating a Java EE Application web application in netbeans IDE, available at https://netbeans.org/kb/docs/javaee/javaee-gettingstarted.html .[Accessed on 23rd February 2018]

[6] The exponential growth of data available at https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/ .[Accessed on 23rd February 2018]

[7] Global internet user penetration . available at https://www.statista.com/statistics/325706/global-internet-user-penetration/ .[Accessed on 23rd February 2018]

[8] Global internet user penetration . available at https://www.statista.com/statistics/325706/global-internet-user-penetration/ .[Accessed on 23rd February 2018]

[9] Binary tree visualizer . available at http://btv.melezinek.cz/ .[Accessed on 23rd February 2018]

Appendices

Survey questionnaire and respondent's feedback

This short survey is part of my bachelor thesis which is about how to design efficient systems, especially in this time where the number of applications users keeps increasing. Please do share with me your views by filling in the following short questionnaire. Thank you in advance for helping out with your feedback.

Jean Bertrand Habinshuti

1. What is your gender?

- Female
- Male

2. Which domain of software are you mostly involved in?

- Software engineering
- Web development
- Application development
- Software testing
- Other

3. How often do you get involved in creating software(or a piece of software)?

- Very often(For example i work full time in a software company)
- Often(For example i work part time in a software company)
- Occasionally(For example only when i have a project at university)

4. For whom do you create a software?

- Real world users(For example ,i work for a company that make software products for end users)
- For school(For example ,semester project)
- Just for fun

5. What do you focus on the most when creating a software(a piece of software)?

- The efficiency(software should accomplish the task in as short period of time as possible)
- The layout(arrangement of parts of a software)
- I don't care, as long as the task is accomplished

6. Between efficiency and effectiveness, which one do you think is more important in software design?

- Efficiency(considering the resources such as time and memory)
- Effectiveness(only focusing on results without regards to resources used)
- It does not matter

* 7. Explain your choice in question number 6

Showing **10** responses

Depending on the amount of users expected to be using our products and how much data our applications will contain, in my company we always try to find ways to make things run smoothly.Offcourse this means that most of our projects take long time to be accomplished but in the end it pays off.

3/4/2018 10:29 AM View respondent's answers

The projects i make are just simple ones so i really do not care.

3/3/2018 6:36 PM View respondent's answers

I do not care much about efficiency.Most of the times i am only doing web development as a way of putting in practice what i have learned in different tutorials,so as long as the end result is good that's what matters to me.I think efficiency should be taken into account by professionals making products for real end users That's my view

3/3/2018 4:48 AM View respondent's answers

Minimally, all use cases need to be covered with satisfactory precision. I consider this primary to efficiency from the viewpoint of a (lazy) developer.

3/1/2018 1:30 PM View respondent's answers

Because

Showing **10** responses

Hard to say. It depends on the system and for whom it is creating..

2/28/2018 9:29 PM View respondent's answers

efficiency is much more important, because, if you have effecient (and nicely written) piece of software, there is almost an implication, that it will be effective at what it is suppoused to do

2/28/2018 7:07 PM View respondent's answers

u have to take care about resources

2/28/2018 6:39 PM View respondent's answers

It all depends on the number of users expected to access the appllication. If the mumber is high,it is necessary to focuss on efficiency otherwise ut does nit really matter

2/27/2018 4:19 PM View respondent's answers

Because by focusing on efficiency , we can save some memory. Also applications become more usable when they take short time to accomplish the tasks.

2/27/2018 4:08 PM View respondent's answers