

### SIX WEEKS SUMMER TRAINING

**“REPORT”**

ON

### DSA Self-Paced (GeeksForGeeks)

Submitted by

# Sumit Kumar Bhagat Registration no:12012276

# Program name: B.Tech(CSE)

*Under the guidance of* “**Sandeep Jain**”

**School of Computer Science & Engineering Lovely Professional University, Phagwara**

# DECLARATION

I,Sumit Kumar Bhagat ,12012276 hereby declare that I have completed my six weeks summer training at GeeksForGeeks from 20 May 2022 to 20 July 2022 under the guidance of Sandeep Jain. I declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of DSA (self-paced), Lovely Professional University, Phagwara.

Date-20.07.2022

Text, whiteboard

Description automatically generated

(Signature of student)

Name of Student Sumit Bhagat

Registration no: 12012276

**ACKNOWLEDGEMENT**

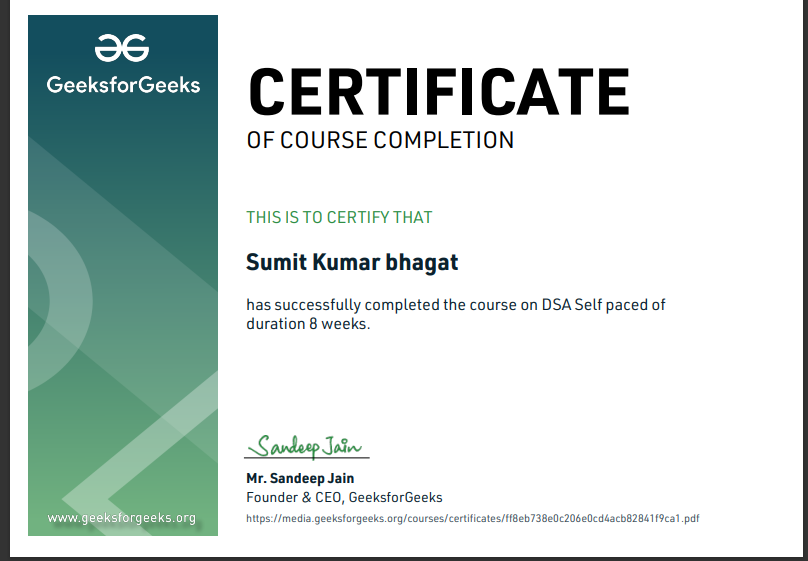
I would like to express my gratitude towards my University as well as Geeks for Geeks for providing me the golden opportunity to do this wonderful summer training regarding DSA, which also helped me in doing a lot of homework and learning. As a result, I came to know about so many new things. So, I am really thank full to them.

Moreover I would like to thank my friends who helped me a lot whenever I got stuck in some problem related to my course. I am really thankfull to have such a good support of them as they always have my back whenever I need.

Also,I would like to mention the support system and consideration of my parents who have always been there in my life to make me choose right thing and oppose the wrong. Without them I could never had learned and became a person who I am now.

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

# CERTIFICATE

****

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| S.No. | Title | Page No. |
| 1. | Introduction | 6 |
| 2. | Technology Learnt | 7-25 |
| 3. | Reason for choosing DSA | 26 |
| 4. | Learning outcome | 27-29 |
| 5. | Project | 30-35 |
| 6. | Bibliography | 36 |

# INTRODUCTION

What is Data Structure? Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.

What is Algorithm? An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain predefined task. Algorithm is not the complete code or program, it is just the core logic(solution) of a problem, which can be expressed either as an informal high- level description as pseudocode or using a flowchart

DSA self paced course is a complete package that helped me to learn Data Structures and Algorithms from Basic to an Advance level. The course curriculm has been divided into 10 weeks, where I practiced questions and I have attempted the assesment tests accordingly. The course offers a wealth of programming challenges that helped me to learn all about DSA and making of an algorithm and how to solve problems and the logic behind the Algorithm.

The course was Self placed means I could join the course anytime and all the content will be avilable to me once I get enrolled. There was video lectures to learn form and multiple choice questions to practice.

I learned Algorithmic techniques for solving various problems with full flexibility of time as I was not time bounded.

This course does not require any prior knowledge of Data Structure and Algorithms, but a basic knowledge of any programming language (C++ / Java) will be helpful.

And as we all know Data Structure and Algorithm is a must skill in terms of Placement in any company because it helps us to increase our problem-solving skill.

# Technology Learnt

#### It had 24 units which was further divided into chapters and then topics so during my whole 10 week course I learned the following :

**INTRODUCTION TO DSA**

#### Analysis of Algorithm

* + In this I learned about background analysis through a Program and its functions.

#### Order of Growth

* + A mathematical explanation of the growth analysis through limits and functions.
  + A direct way of calculating the order of growth

#### Asymptotic Notations

* + Best, Average and Worst-case explanation through a program.

#### Big O Notation

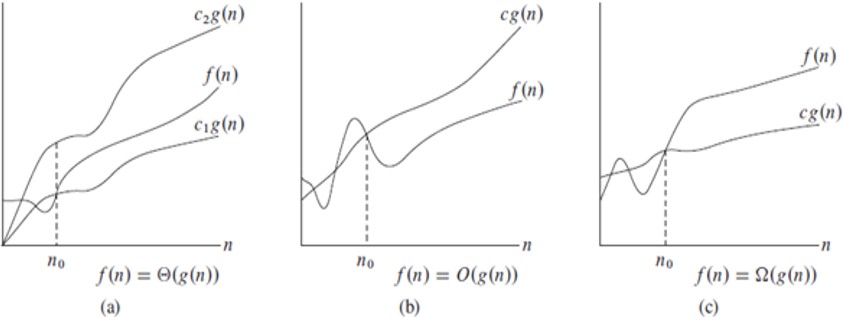
* + Graphical and mathematical explanation.
  + Calculation
  + Applications at Linear Search

#### Omega Notation

* + Graphical and mathematical explanation.
  + Calculation.

#### Theta Notation

* + Graphical and mathematical explanation.
  + Calculation.



#### Analysis of common loops

* + Single, multiple, and nested loops

#### Analysis of Recursion

* + Various calculations through Recursion Tree method

#### Space Complexity

* + Basic Programs
  + Auxiliary Space
  + Space Analysis of Recursion
  + Space Analysis of Fibonacci number

#### MATHEMATICS

* **Finding the number of digits in a number.**

#### Arithmetic and Geometric Progressions.

* **Quadratic Equations.**

#### Mean and Median.

* **Prime Numbers.**

#### LCM and HCF

* **Factorials**

#### Permutations and Combinations

* **Modular Arithmetic**

#### BITMAGIC

* **Bitwise Operators in C++**
  + Operation of AND, OR XOR operators
  + Operation of Left Shift, Right Shift and Bitwise Not

#### Bitwise Operators in Java

* + Operation of AND, OR
  + Operation of Bitwise Not, Left Shift
  + Operation of Right Shift and unsigned Right Shift

#### Problem (With Video Solutions): Check Kth bit is set or not

* + Method 1: Using the left Shift.
  + Method 2: Using the right shift

#### RECURSION

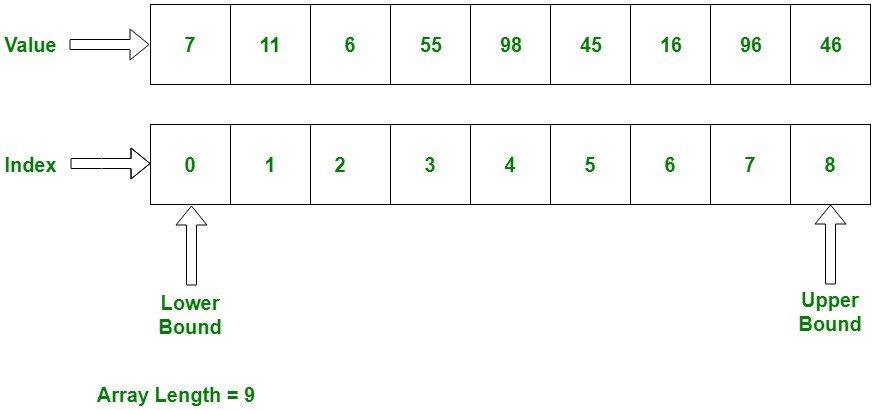
* **Introduction to Recursion**

#### Applications of Recursion

* **Writing base cases in Recursion**
  + Factorial
  + Nth Fibonacci number

#### ARRAYS

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). The base value is index 0 and the difference between the two indexes is the offset.



#### Introduction and Advantages

* **Types of Arrays**
  + Fixed-sized array
  + Dynamic-sized array

#### Operations on Arrays

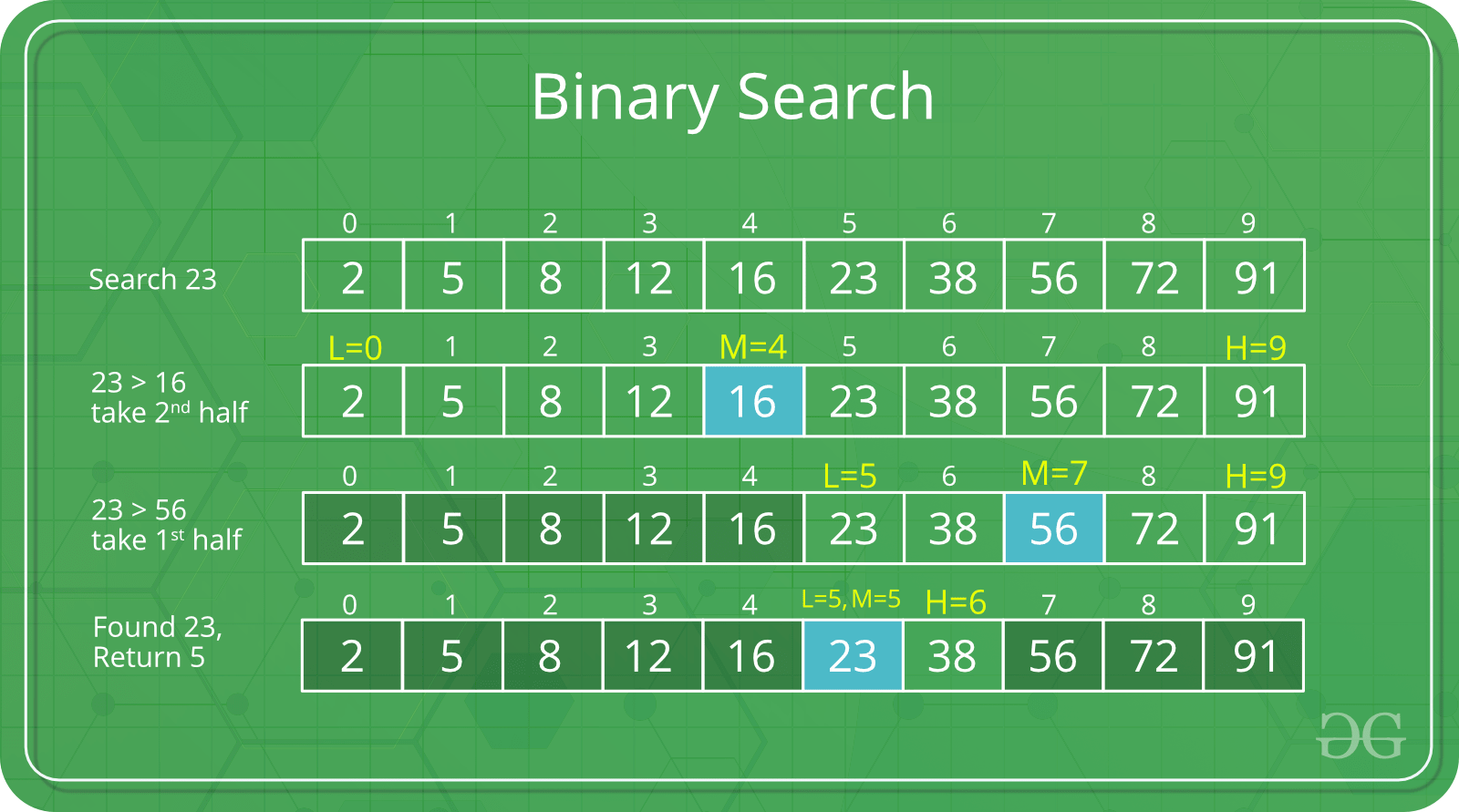
* + Searching
  + Insertions
  + Deletion
  + Arrays vs other DS
  + Reversing - Explanation with complexity

#### SEARCHING

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

#### Binary Search Iterative and Recursive

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found, or the interval is empty.



#### Binary Search and various associated problems

* **Two Pointer Approach Problems SORTING**

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

#### Implementation of C++ STL sort () function in Arrays and Vectors

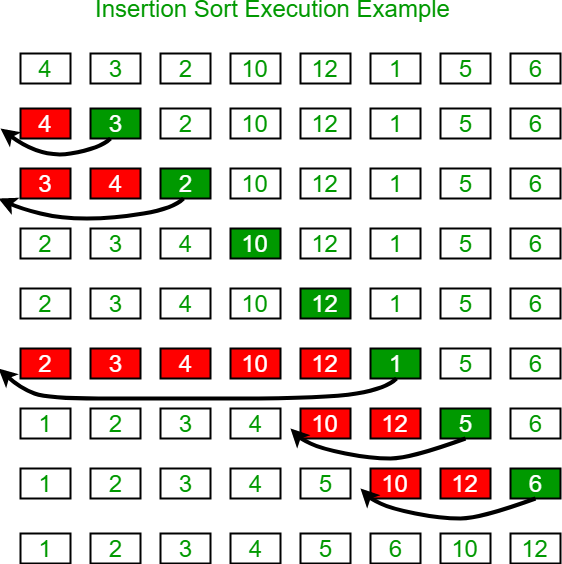
* + Time Complexities

#### Stability in Sorting Algorithms

* + Examples of Stable and Unstable Algos

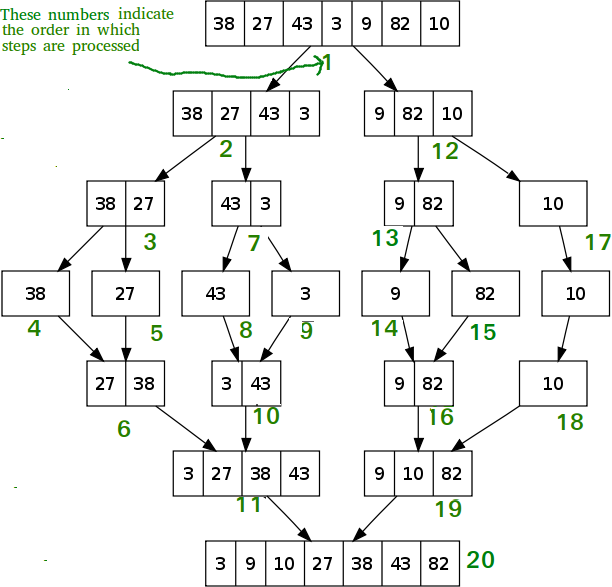
#### Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.



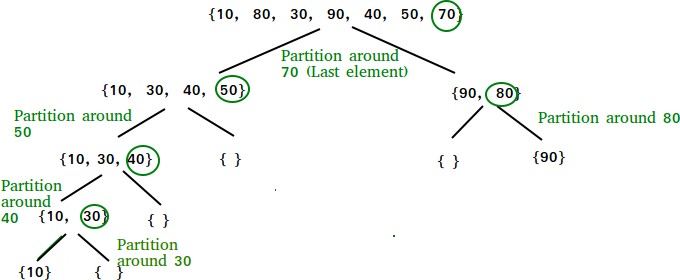
#### Merge Sort

It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. **The merge () function** is used for merging two halves. The merge (arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.



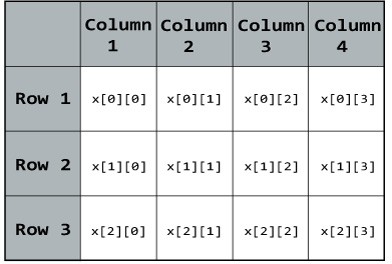
#### Quick Sort

Quicksort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.



* + Using Lomuto and Hoare
  + Time and Space analysis
  + Choice of Pivot and Worst case

#### Overview of Sorting Algorithms MATRIX



* **Introduction to Matrix in C++ and Java**

#### Multidimensional Matrix

* **Pass Matrix as Argument**

#### Printing matrix in a snake pattern

* **Transposing a matrix**

#### Rotating a Matrix

* **Check if the element is present in a row and column-wise sorted matrix.**

#### Boundary Traversal

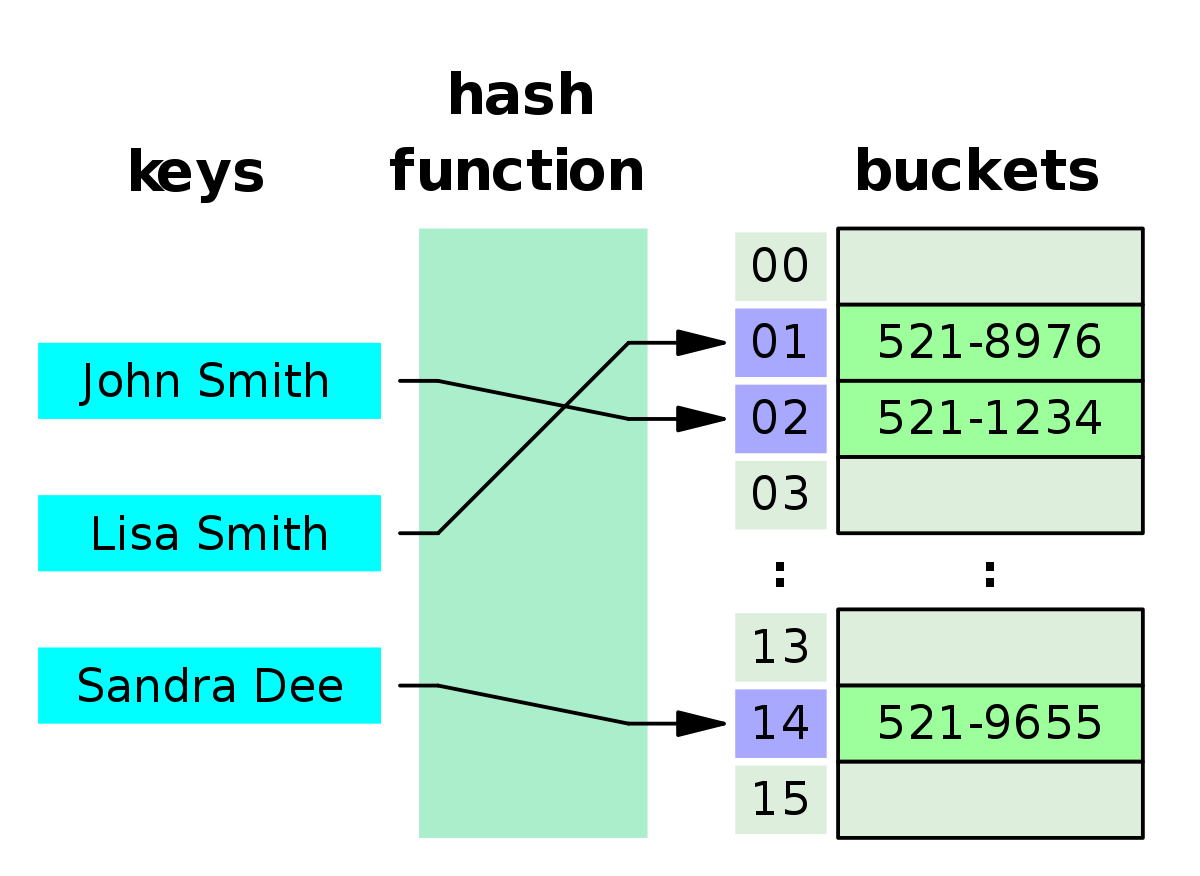
* **Spiral Traversal**

#### Matrix Multiplication

* **Search in row-wise and column-wise Sorted Matrix**

#### HASHING

Hashing is **the process of converting a given key into another smaller value for O(1) retrieval time**. This is done by taking the help of some function or algorithm which is called as hash function to map data to some encrypted or simplified representative value which is termed as “hash code” or “hash”.



#### Introduction and Time complexity analysis

* **Application of Hashing**

#### Discussion on Direct Address Table

* **Working and examples on various Hash Functions**

#### Introduction and Various techniques on Collision Handling

* **Chaining and its implementation**

#### Open Addressing and its Implementation

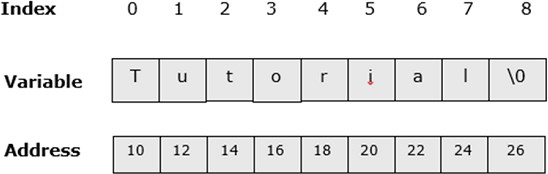
* **Chaining V/S Open Addressing**

#### Double Hashing

* **C++**
  + Unordered Set
  + Unordered Map

#### STRINGS

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character ‘\0’.



#### Discussion of String DS

* **Strings in CPP**

#### Rabin Karp Algorithm

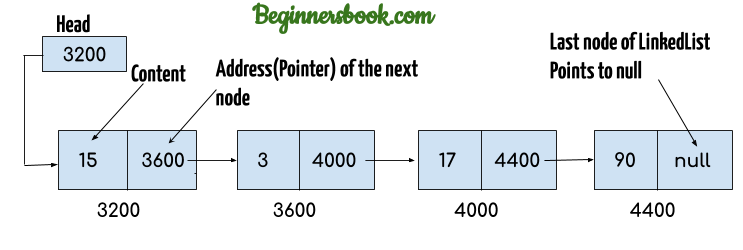
The Rabin-Karp string matching algorithm calculates a hash value for the pattern, as well as for each M-character subsequences of text to be compared. If the hash values are unequal, the algorithm will determine the hash value for next M-character sequence. If the hash values are equal, the algorithm will analyse the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and character matching is only required when the hash values match.

#### KMP Algorithm

Knuth-Morris and Pratt introduce a linear time algorithm for the string-matching problem. A matching time of O (n) is achieved by avoiding comparison with an element of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs.

#### LINKED LIST

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

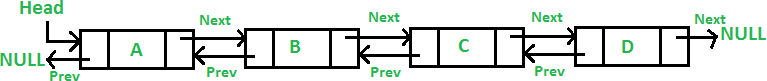


#### Introduction

* + Implementation in CPP
  + Implementation in Java
  + Comparison with Array DS

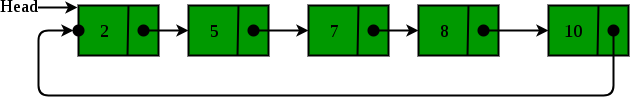
#### Doubly Linked List

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.



#### Circular Linked List

**Circular linked list** is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

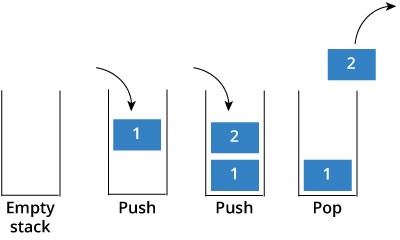


#### Loop Problems

* + Detecting Loops
  + Detecting loops using Floyd cycle detection
  + Detecting and Removing Loops in Linked List

#### STACK

A stack is **an abstract data type that holds an ordered, linear sequence of items**. In contrast to a queue, a stack is a last in, first out (LIFO) structure. A real-life example is a stack of plates: you can only take a plate from the top of the stack, and you can only add a plate to the top of the stack.



#### Understanding the Stack data structure

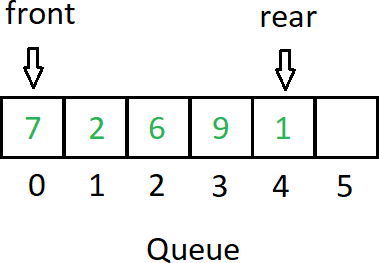
* **Applications of Stack**

#### Implementation of Stack in Array and Linked List

* + In C++
  + In Java

#### QUEUE

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).

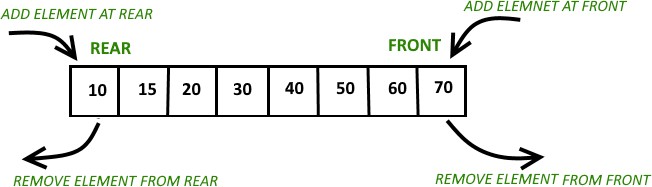


#### Introduction and Application

* **Implementation of the queue using array and LinkedList**
  + In C++ STL
  + Stack using queue

#### DEQUE

A deque, also known as a double-ended queue, is **an ordered collection of items similar to the queue**. It has two ends, a front and a rear, and the items remain positioned in the collection. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.



#### Introduction and Application

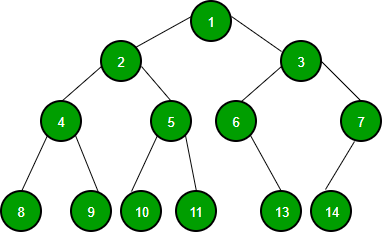
* **Implementation**
  + In C++ STL

#### Problems (With Video Solutions)

* + Maximums of all subarrays of size k
  + ArrayDeque in Java
  + Design a DS with min max operations

#### TREE

A tree is **a nonlinear data structure**, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes, or a tree is a structure consisting of one node called the root and zero or one or more subtrees.



#### Introduction

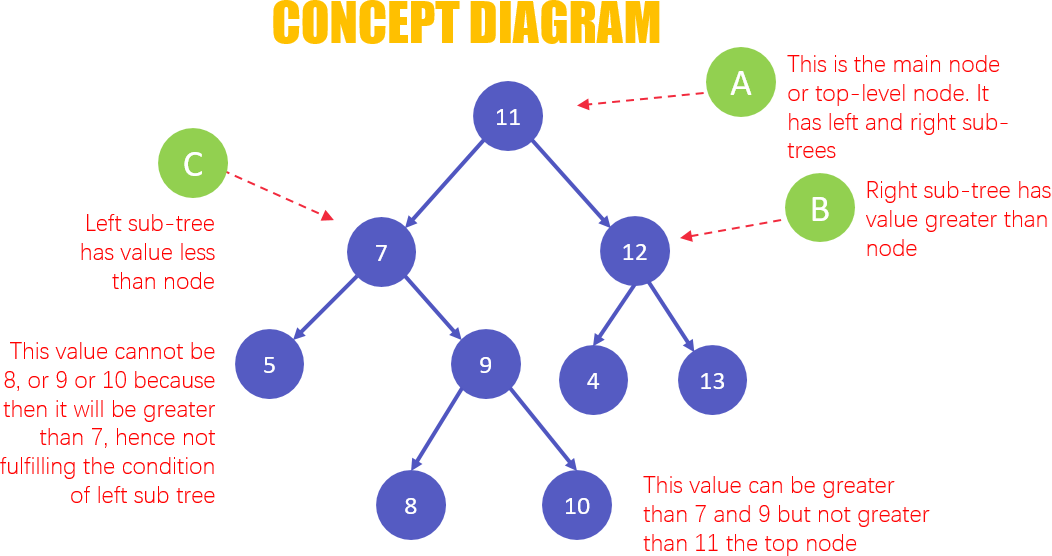
* + Tree
  + Application
  + Binary Tree
  + Tree Traversal

#### Implementation of:

* + Inorder Traversal
  + Preorder Traversal
  + Postorder Traversal
  + Level Order Traversal (Line by Line)
  + Tree Traversal in Spiral Form

#### BINARY SEARCH TREE

Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.



#### Background, Introduction and Application

* **Implementation of Search in BST**

#### Insertion in BST

* **Deletion in BST**

#### Floor in BST

* **Self Balancing BST**

#### AVL Tree

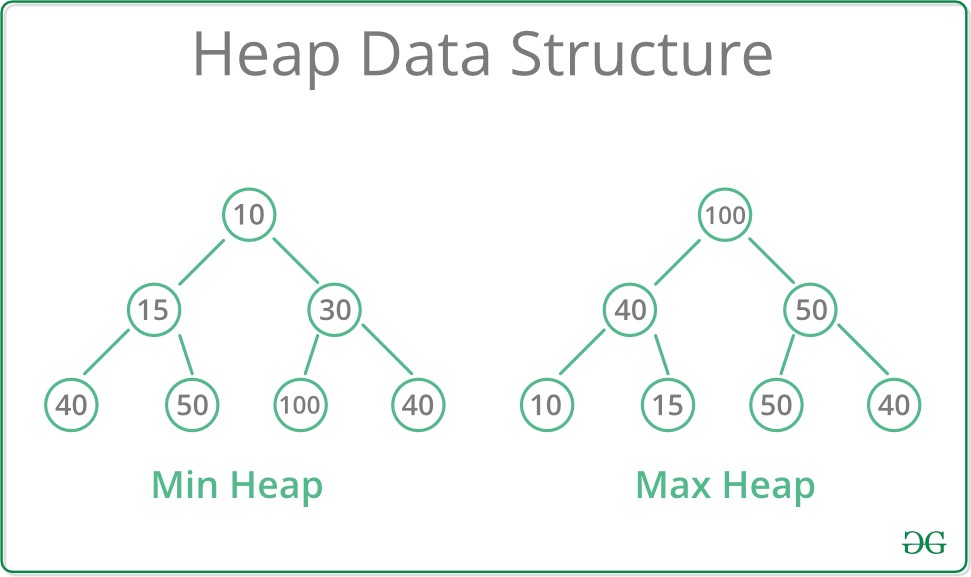
* **Red Black Tree**

#### Set in C++ STL

* **Map in C++ STL**

#### HEAP

A Heap is a special Tree-based data structure in which the tree is a complete binary tree.



#### Introduction & Implementation

* **Binary Heap**
  + Insertion
  + Heapify and Extract
  + Decrease Key, Delete and Build Heap

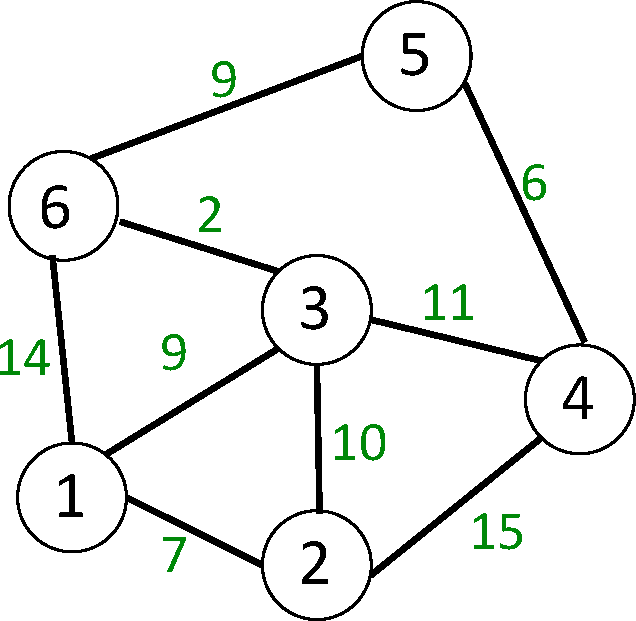
#### Heap Sort

* **Priority Queue in C++**

#### GRAPH

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as,

A Graph consists of a finite set of vertices (or nodes) and set of Edges which connect a pair of nodes.



#### Introduction to Graph

* **Graph Representation**
  + Adjacency Matrix
  + Adjacency List in CPP and Java
  + Adjacency Matrix VS List

#### Breadth-First Search

* + Applications

#### Depth First Search

* + Applications

#### Shortest Path in Directed Acyclic Graph

* **Prim's Algorithm/Minimum Spanning Tree**

#### Dijkstra's Shortest Path Algorithm

* **Bellman-Ford Shortest Path Algorithm**

#### Kosaraju's Algorithm

* **Articulation Point**

#### Bridges in Graph

* **Tarjan’s Algorithm GREEDY**

#### Introduction

* **Activity Selection Problem**

#### Fractional Knapsack

* **Job Sequencing Problem**

#### BACKTRACKING

* **Concepts of Backtracking**

#### Rat In a Maze

* **N Queen Problem**

#### DYNAMIC PROGRAMMING

* **Introduction**

#### Dynamic Programming

* + Memoization
  + Tabulation

#### TREE

* **Introduction**
  + Representation
  + Search
  + Insert
  + Delete

#### Count Distinct Rows in a Binary Matrix

**SEGMENT TREE**

#### Introduction

* **Construction**

#### Range Query

* **Update Query**

#### DISJOINT SET

* **Introduction**

#### Find and Union Operations

* **Union by Rank**

#### Path Compression

* **Kruskal's Algorithm**
  + Improved my problem-solving skills by practicing problems to become a stronger developer
  + Developed my analytical skills on Data Structures to use them efficiently
  + Solved problems asked in product-based companies’ interviews
  + Solved problems in contests similar to coding round for SDE role

# Reason for choosing DSA

* With advancement and innovation in technology, programming is becoming a highly in-demand skill for Software Developers. Everything you see around yourself from Smart TVs, ACs, Lights, Traffic Signals uses some kind of programming for executing user commands.

**Data Structures** and **Algorithms** are the identity of a good Software Developer. The interviews for technical roles in some of the tech giants like *Google, Facebook, Amazon, Flipkart* is more focused on measuring the knowledge of Data Structures and Algorithms of the candidates. The main reason behind this is Data Structures and Algorithms improves the problem-solving ability of a candidate to a great extent.

* This course has video lectures of all the topics from which one can easily learn. I prefer learning from video rather than books and notes. I know books and notes and thesis have their own significance but still video lecture or face to face lectures make it easy to understand faster as we are involved Practically.
* It has 200+ algorithmic coding problems with video explained solutions.
* It has track based learning and weekly assessment to test my skills.
* It was a great opportunity for me to invest my time in learning instead of wasting it here and there during my summer break in this Covid-19 pandemic.
* This was a lifetime accessible course which I can use to learn even after my training whenever I want to revise.

# Learning Outcomes

Programming is all about data structures and algorithms. Data structures are used to hold data while algorithms are used to solve the problem using that data.

Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

For example, you want to search your roll number in 30000 pages of documents, for that you have choices like Linear search, Binary search, etc. So, the more efficient way will be Binary search for searching something in a huge amount of data.

So, if you know the DSA, you can solve any problem efficiently. The main use of DSA is to make your code scalable because

* Time is precious
* Memory is expensive

In our daily life, we always go with that person who can complete the task in a short amount of time with efficiency and using fewer resources. The same things happen with these companies. The problem faced by these companies is much harder and at a much larger scale. Software developers also must make the right decisions when it comes to solving the problems of these companies.

Knowledge of data structures like Hash Tables, Trees, Tries, Graphs, and various algorithms goes a long way in solving these problems efficiently and the interviewers are more interested in seeing how candidates use these tools to solve a problem. Just like a car mechanic needs the right tool to fix a car and make it run properly, a programmer needs the right tool (algorithm and data structure) to make the software run properly. So, the interviewer wants to find a candidate who can apply the right set of tools to solve the given problem. If you know the characteristics of one data structure in contrast to another you will be able to make the right decision in choosing the right data structure to solve a problem.

### Another use of DSA, if you love to solve the real-world complex problems.

Let’s take the example of Library. If you need to find a book on Set Theory from a library, you will go to the math section first, then the Set Theory section. If these books are not organized in this manner and just distributed randomly then it will be frustrating to find a specific book. So, data structures refer to the way we organize information on our computer. Computer scientists process and look for the best way we can organize the data we have, so it can be better processed based on input provided.

A lot of newbie programmers have this question that where we use all the stuff of data structure and algorithm in our daily life and how it’s useful in solving the real-world complex problem. We need to mention that whether you are interested in getting into the top tech giant companies or not DSA still helps a lot in your day-to-day life.

Let’s consider some examples

* In Facebook you can represent your friends on Facebook, friends of friends, mutual friends easily by Graph. 21
* If you need to keep a deck of cards and arrange it properly, how would you do that? You will throw it randomly or you will arrange the cards one over another and from a proper deck. You can use Stack here to make a proper arrangement of cards one over another.
* If you need to search a word in the dictionary, what would be your approach? Do you go page by page or you open some page and if the word is not found you open a page prior/later to one opened depending upon the order of word to the current page (Binary Search).

The first two were a good example of choosing the right data structure for a real-world problem and the third one is a good example of choosing the right algorithm to solve a specific problem in less amount of time.

#### WHAT I LEARNED FROM THE COURSE PRECISELY:

* I Learned Data Structures and Algorithms from basic to advanced level.
* Learned Topic-wise implementation of different Data Structures & Algorithms.
* Improved my problem-solving skills to become a stronger developer.
* Developed my analytical skills on Data Structures and use them efficiently.
* Solved problems asked in product-based companies’ interviews.
* Solved problems in contests similar to coding round for SDE role.

This will help me during my career as a programmer and afterwards also whenever I need to code. We are surrounded by a lot of real-world complex problems for which no one has the solution. Observe the problems in-depth and you can help this world giving the solution which no one has given before.

***“Data structure and algorithms help in understanding the nature of the problem at a deeper level and thereby a better understanding of the world.”***

# Project

**Name of Project -Word counter**

Programme is about a simple console application which reads a text file which can be piece of a novel, newsletter etc. Example data files will be in OIS. Text files will be simple like (much longer though):

--It was many and many a year ago, In a kingdom by the sea,

That a maiden there lived whom you may know By the name of Annabel lee.

And this maiden she lived with no other thought Than to love and be loved by me.

I was a child, and she was a child, In this kingdom by the sea.

……..

--

Our program must be able to read individual words from such files and insert it into a circular singly linked list, if it is a new word. If that word is already encountered before its counter must be increased by 1. The list must always be ordered alphabetically. In other words when inserting the new word, you must insert it to the right place, so alphabetical order is not broken.

After reading and processing is over, the program gives following options to the user.

1. List the most popular words
2. List total number of words.
3. Print the counter value of a specific word.

If option a is selected, the program asks how many words the user wants. For example, if user types 5, it will print the most popular 5 words in the text. If option b is selected the user will input, the word she would like to be printed. For example, in the above sample text the word “by” occurs 4 times so this must be printed if the user asks the word “by”.

Whole application is implemented with console facilities (i.e., you do not need advanced GUI elements). The project consists of two parts.

1. Implementation of a circular singly linked list class of “persons” with following operations: add to head, remove from head, add to tail, remove from tail, insertOrderly and getNodeByValue. InsertOrderly is used to enter a new element to the proper alphabetical place. getNodeByValue will be used to search the list word the query word, so its counter value can be printed.

This will be a proper C++ class. It can create many instances of this class. (no third- party libraries including C++ STL, Boost etc. is used)

1. The main program itself. The program creates a circular linked list instance, put new words from the text file to the list or increase their counts there. Also, it is able to ask command options from user for the purpose of
   1. printing most popular words
   2. printing the number of words that is encountered,
   3. printing the counter value of a user given word.

**CODES**

The code consists of three cpp files and three header files namely main.cpp, wordcounter.cpp, Dictionary.cpp, Wordcounter.h, Dictionary.h and list.h

For all the source code of the project, kindly refer to the given drive link below: [click here for source codes](https://drive.google.com/drive/folders/1ZyJszqKloNBxDDqg76QL_SjYRm7gENVK?usp=sharing)

**CODE DESCRIPTION**

1. What does program do?

-It prints most popular words (amount of user enters) basic histotrophic way.

-It gives to user numbers of (non-repetition ones) words in the text.

-It gives to user how much word which is entered by user repeated in the text. -It prints all the non-repeated words.

1. What cannot program do?

- *‘s ‘re ‘ve* cannot be become *is, are, have*. For example, *name’s* becomes *names.*

### Main structure

**List.h** contains basic functions.

**Dictionary.h** is derived from list class. It is formed of nodes which are string type distinctly and it has alphabetic order.

**wordCounter.h** includes Dictionary.h, it creates dictionary object and it fixes words and fills that dictionary with fixed words.

1. What are the extra works (bonuses) we have done? In the list class we have destructor.

In the dictionary class we write copy constructer, overloaded assignment operator, and overloaded control operators.

### Explanations of the Headers

* 1. **List Header**

There are struct Node and class List in the List.h. Node is a struct which contains node pointer next, int type counter, and template type data as data members. List is a class which contains Node template pointer head and tail, int type size as data members. It contains basic linked list functions. Basic linked list functions are its member functions. Also, it contains a destructor which deletes the entire nodes if it were not, there would be memory leak.

### Dictionary Header

It includes List.h and it derived from List class. Some functions in the List class are modified for alphabetic order in Dictionary class. There is an important point in this class; destructor is empty but there is no memory leaks since it calls base class’ destructor.

Member functions are:

 **insertOrderly:** It is a public function which takes 1 parameter - const string &

- and it does not return any value. It takes words and adds to dictionary.

 **getNodeByValue:** It is a public function which takes 1 parameter - string & - and its return type is int. It returns counter of the given word. There is an

important point in that function which is that thanks to 3rd if condition in while loop if given word greater than the comparison word that means searched word is not on the list so function returns 0.

 **addToHead:** It is a public function which takes const string and if it is

alphabetically smaller than the head it calls List class’ addToHead function if the new word is equal to current head it will increase head’s counter by 1 if the new word isn’t smaller than the current head it throws exception.

 **addToTail:** It is a public function which takes const string and if it is

alphabetically greater than the tail it calls List class’ addToTail function if the new word is equal to current tail it will increase tail’s counter by 1 if the new word isn’t greater than the current tail it throws exception.

#### Operators are:

 Assignment operator (=)

 Operator equal to (==)

 Operator Not equal to (! =)

### InsertOrderly (Detailed)

There is a loop which controls if temp is not equal to tail every time and it controls every nodes in the dictionary. Also with tail->next==NULL || tail-

>next==tail statement it enters the loop even if there is only one element in the dictionary.

There are 6 cases which are controlled in that function.: -

* + 1. We check whether list is empty or not in the first case. If it is empty, it calls addToHead function to create a new node.
    2. We check the case that given word is already in the list if it does, it increases word’s counter by 1.
    3. We check given word is smaller than the head if it is it calls addToHead function.
    4. We check if list has only one element and given word greater than it. If it is, it calls addToTail.
    5. We check if the word less than next node’s data if it is, it creates a new node and terminate the function.
    6. If the function was not terminated because of the cases in the loop that means the word is equal to tail or greater than tail. We check the sixth if

statement that if it equals to tail, if it is, function increases tail’s counter by 1.

At the end of the function if function is not determined by previous cases, the word must be greater than tail so function calls addToTail function.

We realized that if the 4th case and 5th case were reversed order, there would be such an error because program tries to reach the NULL’s next.

### WordCounter Header

Purpose of WordCounter header is that opening the text file and adding words to the dictionary.

This header includes Dictionary.h. We created a Dictionary object in the WordCounter class.

Data members are:

 string fileName

 Dictionary dic Member functions are:

 **fixWord:** It is a private function which takes 1 parameter -string &- and its return type is string. It fixes the given words.

 **fillDictionary:** It is a private void function which takes no parameters. It firstly checks that the file exists if it is not, it throws an exception otherwise it opens the file read word by word send all words to the fixWord one by one and send fixed word to the dictionary object’s insertOrderly function.

 **getCounter:** It returns counter. It called getNodeByValue with given word.

 **getSize:** It is a public function it basicly return dictionary's size.

 **print:** It prints all the list.

 **listPopular**: It prints most popular words on the list user enters that much.

### fixWord (Detailed)

It is a helper function which does more than one job. It controls every letter of the given word one by one. It first does all the capital letters to lower case meantime it checks the character is a letter or not. And then checks the character is not letter and either first character or last if it is, program deletes character. If it is not the first or last character and it is not a letter, program checks that is the character is ( - ) or ( ‘ ) if it is, program controls the next index if

the next character is a letter program deletes the current character which are ( - ) or ( ‘ ) character (i.e. grown-up becomes “grownup”). If the next character is not a letter than program gets the substring of the word from begin to the current index and sends the substring to the insertOrderly then continues to check other part (i.e., grown--up becomes two words “grown” and “up”). If character is not (-) or (‘) and not letter program ends the substring to the insertOrderly then continues to check other part (i.e. grown?up becomes two words “grow” and “up”). At the end of the function, it returns fixed word or empty string why word might be sent insertOrderly in the loop.

### fillDictionary (Detailed)

It is a helper function. It takes the file name and open the file, if the file cannot be opened it throws an exception. It takes strings between spaces one by one and calls fixWord. If the returned string is not empty it calls insertOrderly, if it is empty that means words already added to the dictionary by insertOrderly. At the end it closes the file.

### listPopular (Detailed)

It first checks that case given number is greater than the size of the list than throw an exception if it isn't function creates an array which size same as the list it sends nodes list to array and sorts with shell sort technique when the sorting finishes function prints array which size of user enters and then deletes the array.

### Explanation of the Main.cpp

Main.cpp file includes WordCounter.h. There is menu function which includes:

1. List the most popular words in the text.
2. Get the total number of words.
3. Get the counter of the given word.
4. Print words.

**q)** Quit.

In the main it creates WordCounter object and send file address then it sends object to menu function to give options to the user.

# BIBLIOGRAPHY

* <https://www.google.com/>
* [GeeksForGeeks](https://www.geeksforgeeks.org/)

## [DSA(Self-Paced) course](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=GeeksforGeeks&utm_medium=Banner&utm_campaign=GFG_Home_Rightbar_DSA_SP)

**THAT’S ALL THANK YOU!**