



L OVELY
P ROFESSIONAL
U NIVERSITY

SIX WEEKS SUMMER TRAINING REPORT

On

DATA STRUCTURES and ALGORITHMS

Submitted by

SUMAN MOOND

Registration No : 11917820

Programme Name : B.Tech. CSE

Under the Guidance of

GeeksforGeeks Official Instructor Mr. Sandeep Jain Sir

School of Computer Science & Engineering

Lovely Professional University, Phagwara

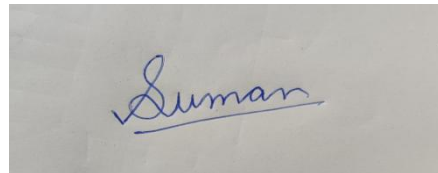
(June-July, 2021)

DECLARATION

I hereby declare that I have completed my six weeks summer training at GeeksForGeeks from 5th June, 2021 to 13th July, 2021 under the guidance of GeeksForGeeks Official Instructor **Mr. Sandeep Jain Sir.**

I have declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of **B.Tech CSE, Lovely Professional University, Phagwara.**

Signature of Student:

A photograph of a handwritten signature in blue ink on a light-colored surface. The signature appears to be 'Suman' with a checkmark at the start and a horizontal line underneath.

Name of Student: **SUMAN MOOND**

Registration no: **11917820**

Date: 5th August, 2021

Summer Training Certificate

The link to the certificate is :

<https://media.geeksforgeeks.org/courses/certificates/53efd6846f7eb18cf9f19fad443fd805.pdf>



Acknowledgement

Firstly, I would like to thank the Department of Computer Science and Engineering of Lovely Professional University, for giving us the opportunity to complete the six weeks summer training on Data Structures and Algorithms. I would also like to thank everyone in the Department for guiding us in completing the course, the mini project and the report. The Department provided us with invaluable advice and helped us in different ways by providing various guidelines and solving multiple issues that arrived in the completing of the summer training, which, in turn motivated me and work with full efforts towards the successful completion of the project.

Besides, we would like to thank all the teachers and the non-teaching staff associated with the Summer Training who helped us by giving us advice and providing us the requisite information that we needed.

Next, I would like to thank the renowned website, “GeeksforGeeks” for making such a useful course, equipping it with video lectures, quizzes, problems and contests. The course was beautifully curated and the instructor, Mr. Sandeep Jain, left no stone unturned in his efforts to make the topics as clear as possible to the students.

Also, I would like to thank my family and friends, and everyone who helped and motivated me to work on the summer training for their support, without which I couldn't have succeeded in completing the training.

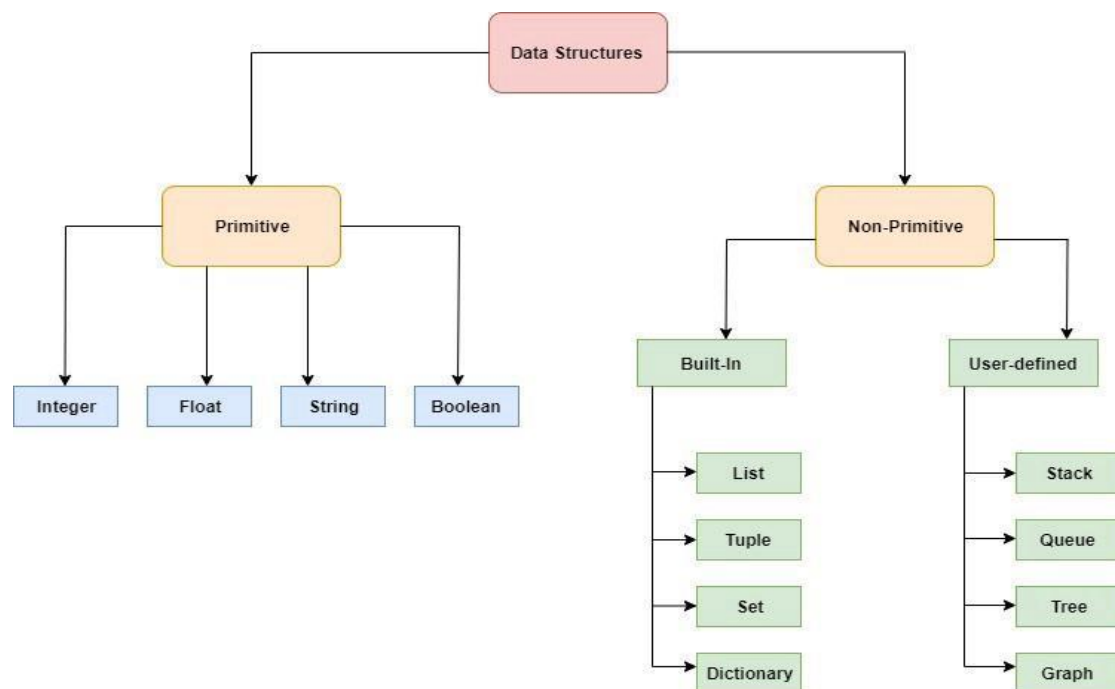
I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

Table of Contents

Serial No.	Title
1.	Declaration By Student
2.	Training Certification from organization
3.	Acknowledgement
4.	Introduction
5.	Project Undertaken
6.	Technology Learn
7.	Reason for Choosing this Technology
8.	Learning Outcome from the Training
9.	Project Legacy
10.	Conclusion
11.	Reference

Introduction

A computer programs is nothing but a collection of instructions that instructs the computer to perform a specific task. To run a program, the processor may need to store, retrieve data and perform operations on the data as instructed by the program written by the developer.



A data structure is a name given to the location that can be used to store and organize data inside the memory of the computer. In short, a data structure is a storage that is used to store and organize data so that it can be accessed and updated efficiently whenever needed by the program runtime.

There are several kinds of data structures and that you can use depending on the requirement of the program.

All the data structures can be primarily divided into 2 parts:

- **Linear Data Structure and**
- **Non-Linear Data Structure**

We can choose between linear and non-linear data structure depending on the type of memory we want. For example, if we want to store data sequentially in the memory, then we can choose the array data structure and implement it in our program. If we don't want a sequential allocation of memory, we can think of linked list.

Let us go through the different types of Data Structures based on their classification as being linear or non-linear:

Linear Data Structures:

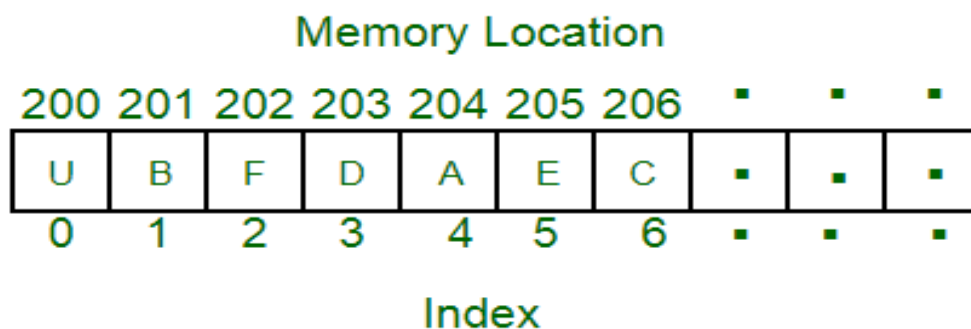
In linear data structures such as arrays and linked lists, the elements to be stored are arranged in sequence one after the other in the memory. Since elements are arranged in particular order, they are easy to implement and are easily accessible to the program.

On the flip side, if the complexity of the program increases, it is better to search the alternatives of the linear data structures.

Some examples of linear data structures are:

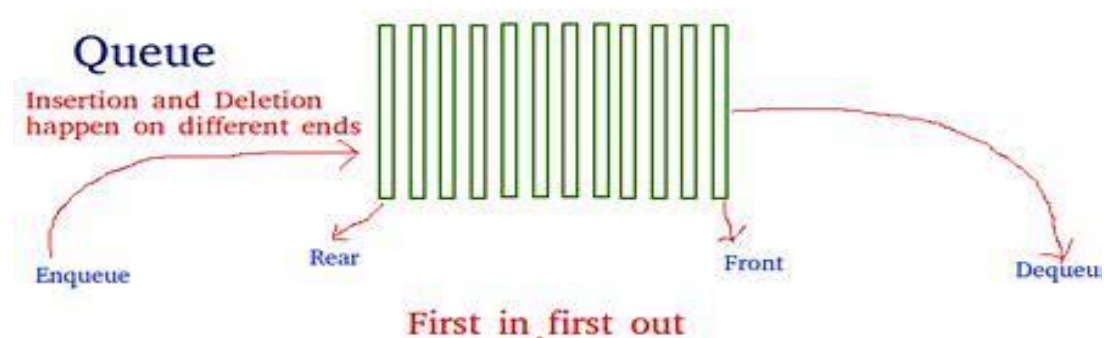
1.)Array Data Structure

An array is the simplest type of linear data structures where the elements in memory are arranged in continuous memory blocks, one next to the other. An important thing to note is that all the elements of an array are of the same type and can comprise of types such as float, int, char, double and so on.



2.)Queue Data Structure

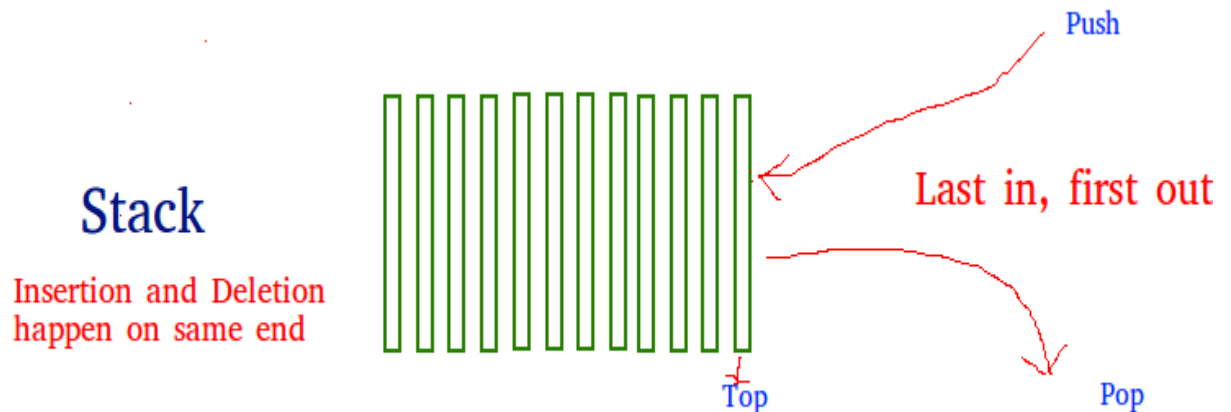
The second type of linear data structure that we will be discussing Queue. Queue data structure works in the FIFO principle which stands for First In First Out. It means that the first element stored in the queue will be removed first from the queue when needed.



It can be visualized by a queue of people in the ticket counter where first person in the queue will get the ticket first when his turn comes

3.)Stack Data Structure

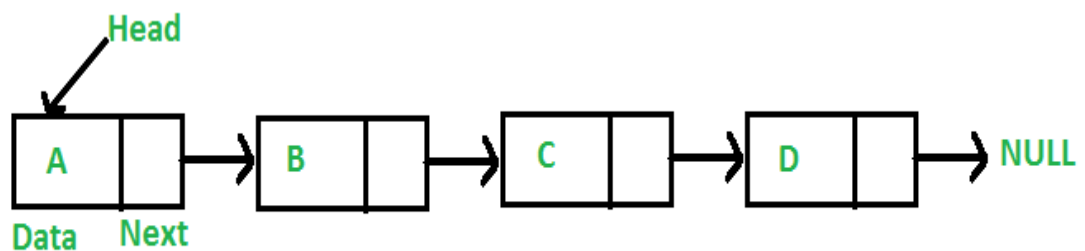
The third type of linear data structure that we will be discussing is the stack data structure where the elements are stored according to the LIFO principle. It stands for Last In First Out which means that the last element stored in a stack will be removed first when time comes to remove the elements from the stack.



It can be visualized as a pile of books where the last book kept on the pile will be removed first when required.

5.)Linked List Data Structure

In linked list data structure, the elements are not stored in contiguous memory location and data elements are connected through a series of nodes. In a basic linked list, each node contains the data items and address to the next node till the end of the linked list which stores NULL.



There are broadly 3 types of linked list:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Non-linear data structures:

In non-linear data structure, elements are not stored in any sequence, and are arranged in a hierarchical manner. It means that in this type of data structure, one element will be connected to one or more elements as per preference.

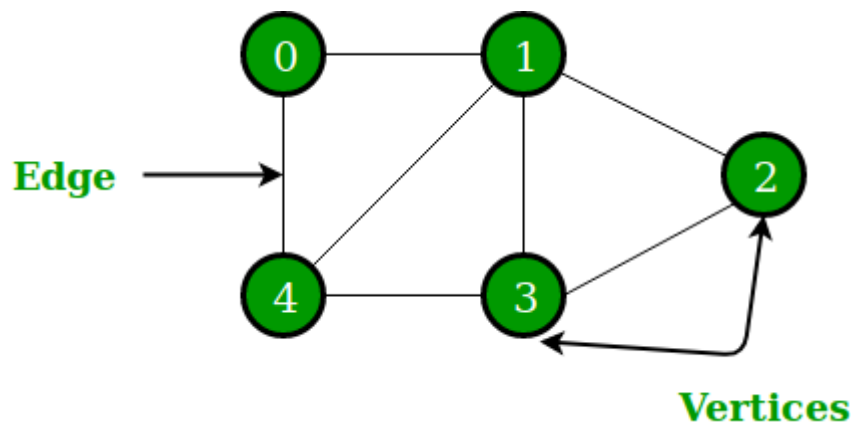
Non-linear data structures are of two types:

- 1. Graph and**
- 2. Tree based data structures.**

Let us go through the different types of Non-Linear Data Structures and have a glance on the overview:

1.)Graph Data Structure

The graph data structure is no what similar to the graphs that we use in mathematics. In graph data structure, each node, also known as the vertex is connected to other vertices through lines known as edges.

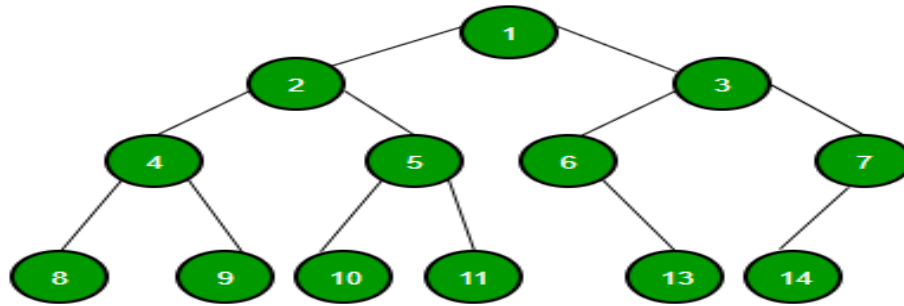


Each vertex can either be connected to only one vertex or more than one vertex through connecting edges. Some examples of graph-based Data Structures are:

- Spanning Tree and Minimum Spanning Tree
- Strongly Connected Components
- Adjacency Matrix
- Adjacency List

2.)Trees Data Structure

A tree is also a collection of vertices and edges, similar to graph with the only difference being only one edge between two vertices.



Some examples of tree-based Data Structures are:

- Binary Tree
- Binary Search Tree
- AVL Tree
- Red-Black Tree

Applications of Data Structure and Algorithms

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms –

- **Search** – Algorithm to search an item in a data structure.
- **Sort** – Algorithm to sort items in a certain order.
- **Insert** – Algorithm to insert item in a data structure.
- **Update** – Algorithm to update an existing item in a data structure.
- **Delete** – Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics –

- **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** – An algorithm should have 0 or more well-defined inputs.
- **Output** – An algorithm should have 1 or more well-defined outputs and should match the desired output.
- **Finiteness** – Algorithms must terminate after a finite number of steps.
- **Feasibility** – Should be feasible with the available resources.
- **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

Usually, the time required by an algorithm falls under three types –

- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution

Complex Types of Data Structure

Then we also have some complex Data Structures, which are used to store large and connected data. Some example of **Abstract Data Structure** are: Linked List, Tree, Graph, Stack, Queue etc.

- Linked List is a kind of data type that stores the data part and a pointer to the next node.

The data is stored in the form of nodes having two parts i.e. data and address part.

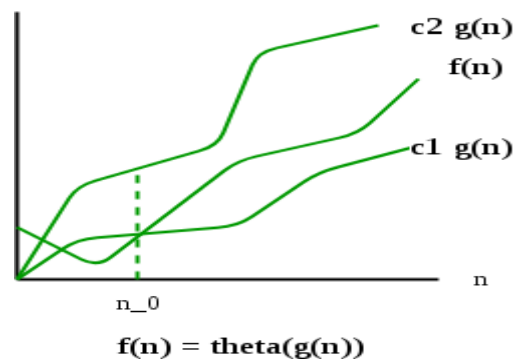
- Tree is a kind of data structure having a root node and multiple child nodes.
- Stack and Queue are the data structures that use LIFO and FIFO principle respectively.

All these data structures allow us to perform different operations on data. We select these data structures based on which type of operation is required.

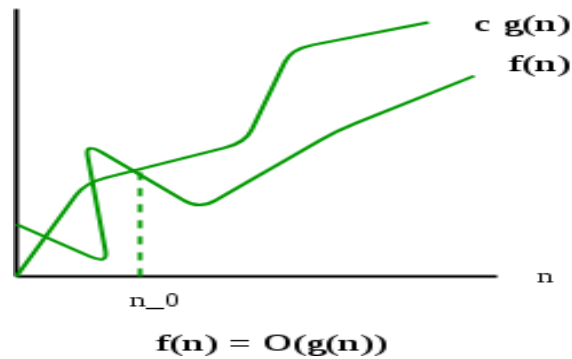
Asymptotic Notation

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. There are three type of asymptotic notations

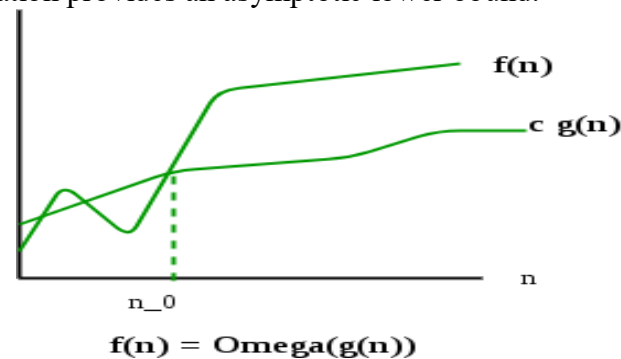
- **Theta notation:** It bounds a function from above and below, so it defines exact asymptotic behavior.



- **Big O Notation:** The Big O notation defines an upper bound of an algorithm, it bounds a function only from above.



- **Ω Notation:** Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.



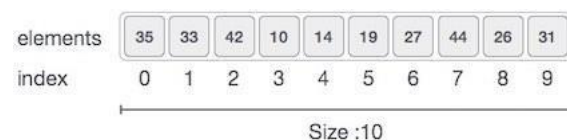
Arrays

An array is a collection of items of the same data type stored at contiguous memory locations. 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.

- **Element** – Each item stored in an array is called an element.
- **Index** – Each location of an element in an array has a numerical index, which is used to identify the element

Array Representation

Arrays can be declared in various ways in different languages.



Index starts with 0.

- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Basic Operations

The basic operations supported by an array.

1. **Traverse** – print all the array elements one by one.
2. **Insertion** – Adds an element at the given index.
3. **Deletion** – Deletes an element at the given index.
4. **Search** – Searches an element using the given index or by the value.
5. **Update** – Updates an element at the given index

Advantages

- It is better and convenient way of storing the data of same datatype with same size.
- It allows us to store known number of elements in it.
- It allocates memory in contiguous memory locations for its elements. It does not allocate any extra space/ memory for its elements. Hence there is no memory overflow or shortage of memory in arrays.
- Iterating the arrays using their index is faster compared to any other methods like linked list etc.
- It allows to store the elements in any dimensional array– supports multidimensional array.

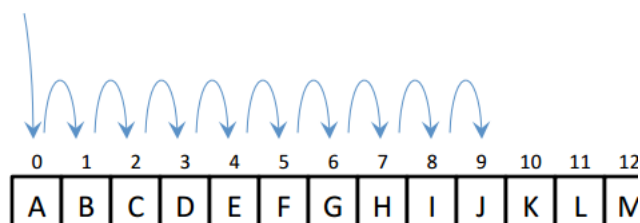
Disadvantages

- It allows us to enter only fixed number of elements into it. We cannot alter the size of the array once array is declared. Hence if we need to insert a greater number of records than declared then it is not possible. We should know array size at the compile time itself.
- Inserting and deleting the records from the array would be costly since we add / delete the elements from the array, we need to manage memory space too.
- It does not verify the indexes while compiling the array. In case there is any indexes pointed which is more than the dimension specified, then we will get run time errors rather than identifying them at compile time.

Searching

Linear Search means to sequentially traverse a given list or array and check if an element is present in the respective array or list. The idea is to start traversing the array and compare elements of the array one by one starting from the first element with the given element until a match is found or end of the array is reached.

Find "J"



Worst case time complexity of the linear search algorithm is $O(N)$.

Binary Search is a searching algorithm for searching an element in a sorted list or array. Binary Search is efficient than Linear Search algorithm and performs the

search operation in logarithmic time complexity for sorted arrays or lists.

If searching for 23 in the 10-element array:

	2	5	8	12	16	23	38	56	72	91
23 > 16, take 2 nd half	L									H
	2	5	8	12	16	23	38	56	72	91
23 < 56, take 1 st half										
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5										
	2	5	8	12	16	23	38	56	72	91

Binary Search performs the search operation by repeatedly dividing the search interval in half. The idea is to 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.

Sorting

Sorting any sequence means to arrange the elements of that sequence according to some specific criterion. For Example,

the array `arr[] = {5, 4, 2, 1, 3}`

after *sorting in increasing order* will be:

`arr[] = {1, 2, 3, 4, 5}`

The same array after *sorting in descending order* will be:

`arr[] = {5, 4, 3, 2, 1}`.

• Insertion Sort

Insertion Sort is an In-Place sorting algorithm. This algorithm works in a similar way of sorting a deck of playing cards. The idea is to start iterating from the second element of array till last element and for every element insert at its correct position in the subarray before it.

• Bubble Sort

Bubble Sort is also an in-place sorting algorithm. This is the simplest sorting algorithm. In one iteration if we swap all adjacent elements of an array such that after swap the first element is less than the second element then at the end of the iteration, the first element of the array will be the minimum element. Bubble-Sort algorithm simply repeats the above steps N-1 times, where N is the size of the array.

• Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- o The subarray which is already sorted.
- o Remaining subarray which is unsorted.

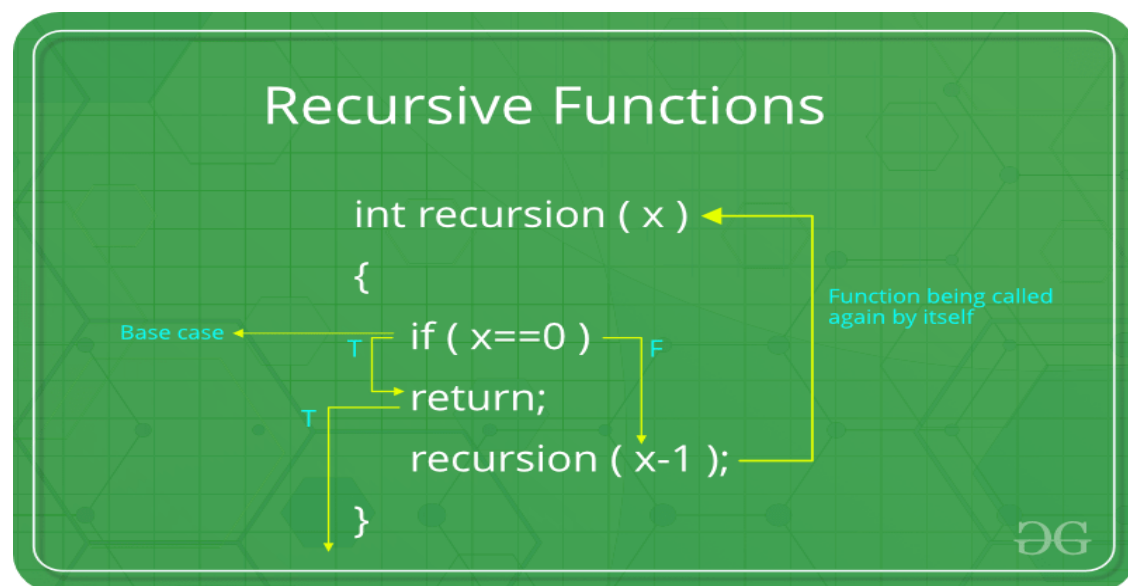
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

RECURISON

Some computer programming languages allow a module or function to call itself. This technique is known as recursion. In recursion, a function α either calls itself directly or calls a function β that in turn calls the original function α . The function α is called recursive function.

Example – a function calling itself



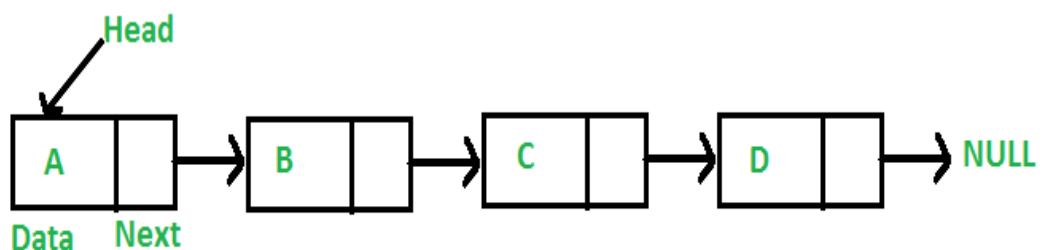
Linked List

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link** – Each link of a linked list can store a data called an element.
- **Next** – Each link of a linked list contains a link to the next link called Next.
- **Linked List** – A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first. Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** – Item navigation is forward only.
- **Doubly Linked List** – Items can be navigated forward and backward.
- **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Advantages of Linked Lists over Arrays

Arrays can be used to store linear data of similar types, but arrays have the following limitations:

1. The size of the arrays is fixed: So, we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage. On the other hand, linked lists are dynamic, and the size of the linked list can be incremented or decremented during runtime.
2. Inserting a new element in an array of elements is expensive, because a room has to be created for the new elements and to create room, existing elements have to shift.

Disadvantages of Linked Lists:

1. Random access is not allowed in Linked Lists. We must access elements sequentially starting from the first node. So, we cannot do a binary search with linked lists efficiently with its default implementation. Therefore, lookup or search operation is costly in linked lists in comparison to arrays.
2. Extra memory space for a pointer is required with each element of the list.
3. Not cache friendly. Since array elements are present at contiguous locations, there is a locality of reference which is not there in case of linked lists.

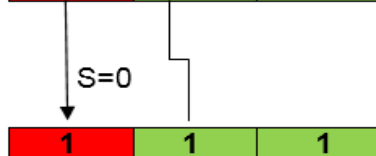
Strings

Strings are defined as a stream of characters. Strings are used to represent text and are generally represented by enclosing text within quotes. In java, objects of String are immutable which means a constant and cannot be changed once created. Some methods in Strings: finding the length and substring and index of string, concatenating two strings, comparing two strings, uppercase, lowercase, trim and replace.

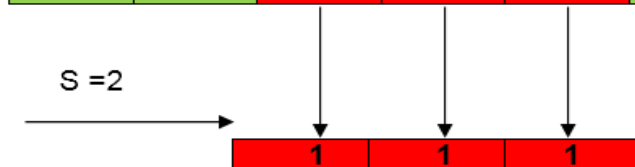
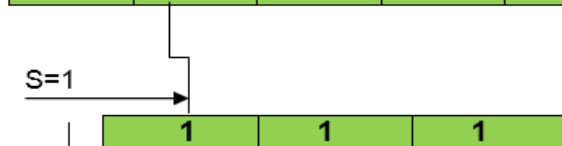
Searching algorithms:

- **Naïve pattern searching:** $O(m*(n-m+1))$
- **Rabin-Karp algorithm:** $O(n+m)$
- **KMP algorithm:** $O(n)$

T = Text



P = Pattern



Stack

The stack is a linear data structure which follows a particular order in which the

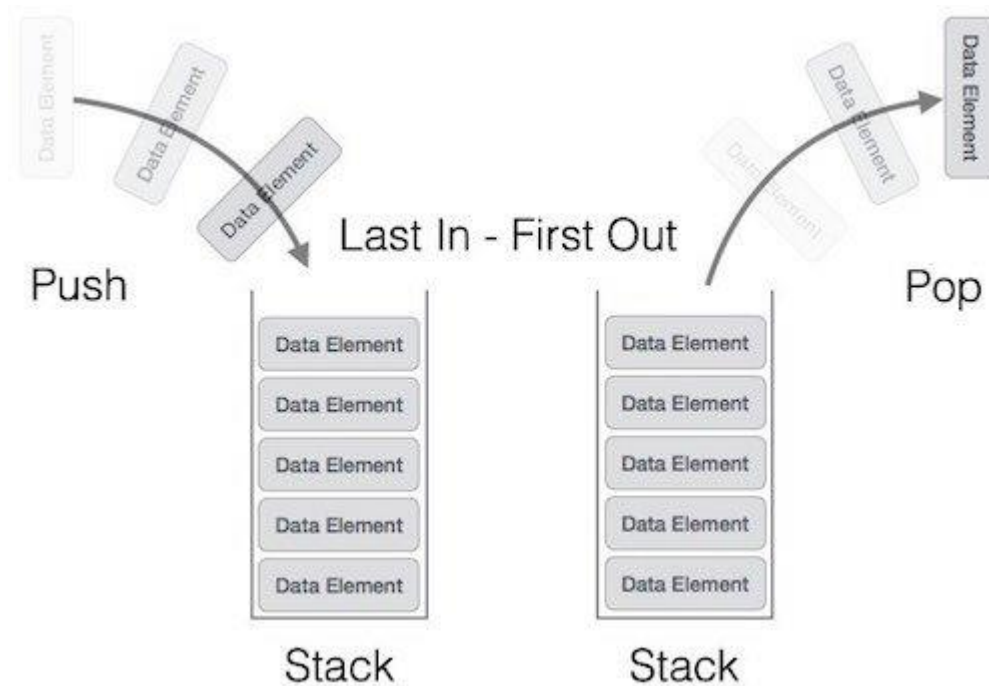
operations are performed. The order may be LIFO (Last in First Out) or FILO (First In Last Out).

The LIFO order says that the element which is inserted at the last in the Stack will be the first one to be removed. In LIFO order insertion takes place at the rear end of the stack and deletion occurs at the front of the stack.

The FILO order says that the element which is inserted at the first in the Stack will be the last one to be removed. In FILO order insertion takes place at the rear end of the stack and deletion occurs at the front of the stack.

Operations: push, pop, peak or Top, is Empty

It can be implemented in two ways using arrays and linked list



Queue

Queue is also a linear data structure which follows a particular order in which the operations are performed.

The order is First in First Out (FIFO) which means that the element which is inserted first in the queue will be the first one to be removed from the queue.

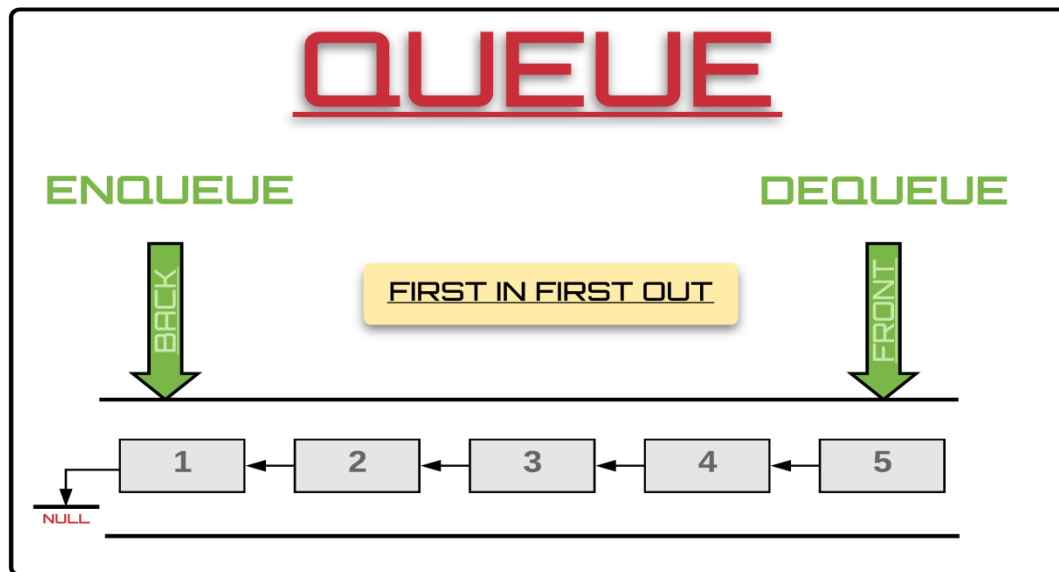
Operations:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue



TREE

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc.) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

Basic terminologies in tree: Root node, leaf node, edge, height of tree

BINARY SEARCH TREE

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child. A Binary Tree node contains following parts.

1. Data
2. Pointer to left child
3. Pointer to right child

Dynamic Programming

Dynamic Programming is an algorithmic approach to solve some complex problems easily and save time and number of comparisons by storing the results of past computations. The basic idea of dynamic programming is to store the results of previous calculation and reuse it in future instead of recalculating them.

There are two main properties of any problem which identifies a problem that it can be solved using the dynamic programming approach:

1. Overlapping Subproblem Property
2. Optimal Substructure Property

Heap

A Heap is a Tree-based data structure, which satisfies the below properties:

1. A Heap is a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible).
2. A Heap is either Min Heap or Max Heap. In a Min-Heap, the key at root must be minimum among all keys present in the Binary Heap. The same property must be recursively true for all nodes in the Tree. Max Heap is similar to MinHeap.

Project Undertaken

Project Name- Number System Conversion

Technology Used: C++

Objectives of the work undertaken

This course is a complete package that helps you learn Data Structures Algorithms from basic to an advanced level. The course offers you programming challenges that will help you to prepare for interviews with top-notch companies.

Scope of the Work

Solve problems asked in product-based companies' interviews.

Solve problems in contests similar to coding round for SDE role.

Importance and Applicability

Improve your problem-solving skills to become a stronger developer.

Develop your analytical skills on Data Structures and use them efficiently.

Role and profile

Learn Data Structures and Algorithms from basic to advanced level

Learn Topic-wise implementation of different Data Structures & Algorithms

Code Snippet:

```
//Number System and Code
#include <iostream>
#include <cmath>
using namespace std;

class con{
public:
    void headline();
    void dec_bin();
    void dec_oct();
    void dec_hex();
    void bin_dec();
```

```

        void bin_oct();
        void bin_hex();
        void oct_dec();
        void oct_bin();
        void oct_hex();
        void hex_dec();
        void hex_bin();
        void hex_oct();
};

int main() {
    con fn;
    fn.headline();
    cout << "***** All in One *****\n";
    cout << "  1. Decimal to Binary\n";
    cout << "  2. Decimal to Octal\n";
    cout << "  3. Decimal to Hexadecimal\n";
    cout << "  4. Binary to Decimal\n";
    cout << "  5. Binary to Octal\n";
    cout << "  6. Binary to Hexadecimal\n";
    cout << "  7. Octal to Decimal\n";
    cout << "  8. Octal to Binary\n";
    cout << "  9. Octal to Hexadecimal\n";
    cout << " 10. Hexadecimal to Decimal\n";
    cout << " 11. Hexadecimal to Binary\n";
    cout << " 12. Hexadecimal to Octal\n";
    cout << " 13. Exit\n";
    cout << "*****\n\n";

    int choice;
    while(choice != 13) {

        cout << "\nEnter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                fn.dec_bin();    break;
            case 2:
                fn.dec_oct();    break;
            case 3:
                fn.dec_hex();    break;
            case 4:
                fn.bin_dec();    break;
            case 5:
                fn.bin_oct();    break;
            case 6:
                fn.bin_hex();    break;
            case 7:

```

```

        fn.oct_dec();    break;
    case 8:
        fn.oct_bin();    break;
    case 9:
        fn.oct_hex();    break;
    case 10:
        fn.hex_dec();    break;
    case 11:
        fn.hex_bin();    break;
    case 12:
        fn.hex_oct();    break;
    case 13:
        cout << "\nYou are logged out Successfully!!\n";
        break;
    default:
        cout << "\nError!!";
    }
}
return 0;
}

void con::headline() {
    cout << "\t\t\t\t\t*****\n";
    cout << "\t\t\t\t\t    Number System & Code\n";
    cout << "\t\t\t\t\t*****\n";
}

void con::dec_bin() {
    int binary[50], dec_num, k=0;

    cout << "Enter the Decimal number : ";
    cin >> dec_num;

    while(dec_num!=1) {
        binary[k] = dec_num % 2;
        dec_num /= 2; k++;
    }
    binary[k] = 1;

    for (int i=k; i>=0; i--) {
        cout << binary[i];
    } cout << endl;
}

void con::dec_oct() {
    int octal[50], dec_num, k=0;

    cout << "Enter the Decimal number : ";
    cin >> dec_num;

```

```

while(dec_num > 8) {
    octal[k] = dec_num % 8;
    dec_num /= 8; k++;
}
octal[k] = dec_num;

for (int i=k; i>=0; i--) {
    cout << octal[i];
} cout << endl;
}

void con::dec_hex() {
    int dec_num, k=0, x[10];
    cout << "Enter the Decimal number : ";
    cin >> dec_num;

    while(dec_num > 16) {
        x[k] = dec_num % 16;
        dec_num /= 16; k++;
    }
    x[k] = dec_num;

    for (int i=k; i>=0; i--) {
        if(x[i]>=10) {
            if(x[i] == 10) cout << 'A';
            if(x[i] == 11) cout << 'B';
            if(x[i] == 12) cout << 'C';
            if(x[i] == 13) cout << 'D';
            if(x[i] == 14) cout << 'E';
            if(x[i] == 15) cout << 'F';
        }else {
            cout << x[i];
        }
    } cout << endl;
}

void con::bin_dec() {
    int decimal=0, k=0;
    unsigned long long int bin_num;
    cout << "Enter a Binary Number : ";
    cin >> bin_num;

    while(bin_num != 0) {
        decimal += (bin_num % 10) * pow(2, k);
        bin_num /= 10;
        k++;
    }
    cout << decimal << endl;
}

```

```

}

void con::bin_oct() {
    //binary >> decimal >> octal
    unsigned long long int bin_num;
    int decimal=0, octal[50], k=0;

    cout << "Enter a Binary Number : ";
    cin >> bin_num;

    while(bin_num != 0) {
        decimal += (bin_num % 10) * pow(2, k);
        bin_num /= 10;
        k++;
    }
    k=0;
    while(decimal > 8) {
        octal[k] = decimal % 8;
        decimal /= 8; k++;
    }
    octal[k] = decimal;

    for (int i=k; i>=0; i--) {
        cout << octal[i];
    } cout << endl;
}

void con::bin_hex() {
    //binary >> decimal >> hexadecimal
    unsigned long long int bin_num;
    int decimal=0, octal[50], x[10], k=0;

    cout << "Enter a Binary Number : ";
    cin >> bin_num;

    while(bin_num != 0) {
        decimal += (bin_num % 10) * pow(2, k);
        bin_num /= 10;
        k++;
    }
    k=0;
    while(decimal > 16) {
        x[k] = decimal % 16;
        decimal /= 16; k++;
    }
    x[k] = decimal;

    for (int i=k; i>=0; i--) {
        if(x[i]>=10) {

```



```

        if(x[i] == 10) cout << 'A';
        if(x[i] == 11) cout << 'B';
        if(x[i] == 12) cout << 'C';
        if(x[i] == 13) cout << 'D';
        if(x[i] == 14) cout << 'E';
        if(x[i] == 15) cout << 'F';
    }else {
        cout << x[i];
    }
} cout << endl;
}

void con::oct_dec() {
    int octal, k=0, decimal=0;
    cout << "Enter a Octal Number : ";
    cin >> octal;

    while(octal != 0) {
        decimal += (octal % 10) * pow(8, k);
        octal /= 10;
        k++;
    }
    cout << decimal << endl;
}

void con::oct_bin() {
    //octal >> decimal >> binary
    int octal, k=0, decimal=0, binary[50];
    cout << "Enter a Octal Number : ";
    cin >> octal;

    while(octal != 0) {
        decimal += (octal % 10) * pow(8, k);
        octal /= 10;
        k++;
    }
    k=0;
    while(decimal!=1) {
        binary[k] = decimal % 2;
        decimal /= 2; k++;
    }
    binary[k] = 1;

    for (int i=k; i>=0; i--) {
        cout << binary[i];
    } cout << endl;
}

void con::oct_hex() {

```

```

//octal >> decimal >> hexadecimal
int octal, k=0, decimal=0, x[10];
cout << "Enter a Octal Number : ";
cin >> octal;

while(octal != 0) {
    decimal += (octal % 10) * pow(8, k);
    octal /= 10;
    k++;
}
k=0;
while(decimal > 16) {
    x[k] = decimal % 16;
    decimal /= 16; k++;
}
x[k] = decimal;

for (int i=k; i>=0; i--) {
    if(x[i]>=10) {
        if(x[i] == 10) cout << 'A';
        if(x[i] == 11) cout << 'B';
        if(x[i] == 12) cout << 'C';
        if(x[i] == 13) cout << 'D';
        if(x[i] == 14) cout << 'E';
        if(x[i] == 15) cout << 'F';
    }else {
        cout << x[i];
    }
} cout << endl;
}

void con::hex_dec() {
    char hex[10];
    int store[10], decimal=0, k=0, i;
    cout << "Enter a Hexadecimal number : ";
    cin >> hex;

    for(i=0; hex[i]!='\0'; i++) {
        if(hex[i]>='A' && hex[i]<='F') {
            if(hex[i] == 'A') store[i] = 10;
            if(hex[i] == 'B') store[i] = 11;
            if(hex[i] == 'C') store[i] = 12;
            if(hex[i] == 'D') store[i] = 13;
            if(hex[i] == 'E') store[i] = 14;
            if(hex[i] == 'F') store[i] = 15;
        }else {
            store[i] = hex[i] - '0';
        }
    }
}

```

```

        for(int j=i-1; j>-1; j--) {
            decimal += store[j] * pow(16, k);
            k++;
        }
        cout << decimal << endl;
    }

void con::hex_bin() {
    //hex >> decimal >> binary
    char hex[10];
    int store[10], decimal=0, k=0, i, binary[50];
    cout << "Enter a Hexadecimal number : ";
    cin >> hex;

    for(i=0; hex[i]!='\0'; i++) {
        if(hex[i]>='A' && hex[i]<='F') {
            if(hex[i] == 'A') store[i] = 10;
            if(hex[i] == 'B') store[i] = 11;
            if(hex[i] == 'C') store[i] = 12;
            if(hex[i] == 'D') store[i] = 13;
            if(hex[i] == 'E') store[i] = 14;
            if(hex[i] == 'F') store[i] = 15;
        }else {
            store[i] = hex[i] - '0';
        }
    }
    for(int j=i-1; j>-1; j--) {
        decimal += store[j] * pow(16, k);
        k++;
    }

    k=0;
    while(decimal!=1) {
        binary[k] = decimal % 2;
        decimal /= 2; k++;
    }
    binary[k] = 1;

    for (int i=k; i>=0; i--) {
        cout << binary[i];
    } cout << endl;
}

void con::hex_oct() {
    //hex >> decimal >> octal
    char hex[10];
    int store[10], decimal=0, k=0, i, octal[50];
    cout << "Enter a Hexadecimal number : ";
    cin >> hex;

```

```

for(i=0; hex[i]!='\0'; i++) {
    if(hex[i]>='A' && hex[i]<='F') {
        if(hex[i] == 'A') store[i] = 10;
        if(hex[i] == 'B') store[i] = 11;
        if(hex[i] == 'C') store[i] = 12;
        if(hex[i] == 'D') store[i] = 13;
        if(hex[i] == 'E') store[i] = 14;
        if(hex[i] == 'F') store[i] = 15;
    }else {
        store[i] = hex[i] - '0';
    }
}
for(int j=i-1; j>-1; j--) {
    decimal += store[j] * pow(16, k);
    k++;
}

k=0;
while(decimal > 8) {
    octal[k] = decimal % 8;
    decimal /= 8; k++;
}
octal[k] = decimal;

for (int i=k; i>=0; i--) {
    cout << octal[i];
} cout << endl;
}

```

```

*****
|                                     |
|      Number System & Code          |
|                                     |
*****
***** All in One *****
1. Decimal to Binary
2. Decimal to Octal
3. Decimal to Hexadecimal
4. Binary to Decimal
5. Binary to Octal
6. Binary to Hexadecimal
7. Octal to Decimal
8. Octal to Binary
9. Octal to Hexadecimal
10. Hexadecimal to Decimal
11. Hexadecimal to Binary
12. Hexadecimal to Octal
13. Exit
*****
|

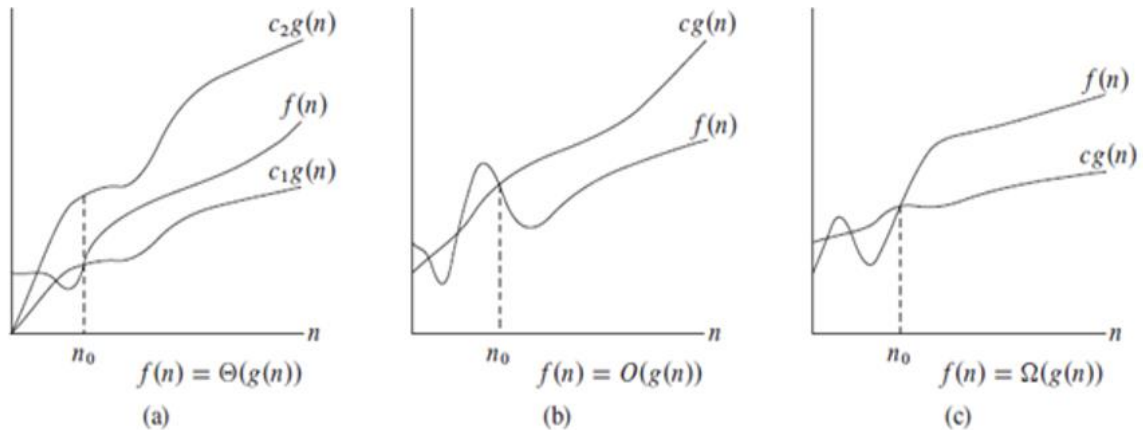
```

Enter your choice:

Technology learnt

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
 - N-th Fibonacci number

ARRAYS

- **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

- **Binary Search Iterative and Recursive**
- **Binary Search and various associated problems**
- **Two Pointer Approach Problems**

SORTING

- **Implementation of C++ STL sort() function in Arrays and Vectors**
 - Time Complexities
- **Sorting in Java**
- **Arrays.sort() in Java**
- **Collection.sort() in Java**
- **Stability in Sorting Algorithms**
 - Examples of Stable and Unstable Algos
- **Insertion Sort**
- **Merge Sort**
- **Quick Sort**

- 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

- **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
- **Java**
 - HashSet
 - HashMap

STRINGS

- **Discussion of String DS**
- **Strings in CPP**
- **Strings in Java**
- **Rabin Karp Algorithm**
- **KMP Algorithm**

LINKED LIST

- **Introduction**
 - Implementation in CPP
 - Implementation in Java
 - Comparison with Array DS
- **Doubly Linked List**
- **Circular Linked List**
- **Loop Problems**

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

STACK

- **Understanding the Stack data structure**
- **Applications of Stack**
- **Implementation of Stack in Array and Linked List**
 - In C++
 - In Java

QUEUE

- **Introduction and Application**
- **Implementation of the queue using array and Linked List**
 - In C++ STL
 - In Java
 - Stack using queue

DEQUE

- **Introduction and Application**
- **Implementation**
 - In C++ STL
 - In Java
- **Problems(With Video Solutions)**
 - Maximums of all subarrays of size k
 - ArrayDeque in Java

- Design a DS with min max operations

TREE

- **Introduction**

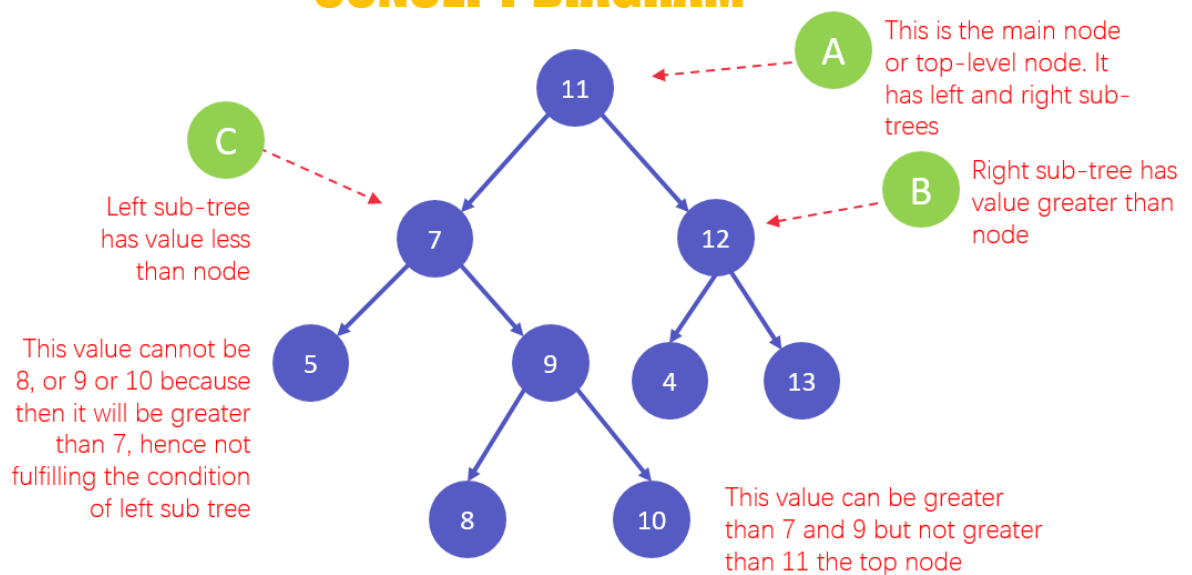
- Tree
- Application
- Binary Tree
- Tree Traversal

- **Implementation of:**

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

BINARY SEARCH TREE

CONCEPT DIAGRAM



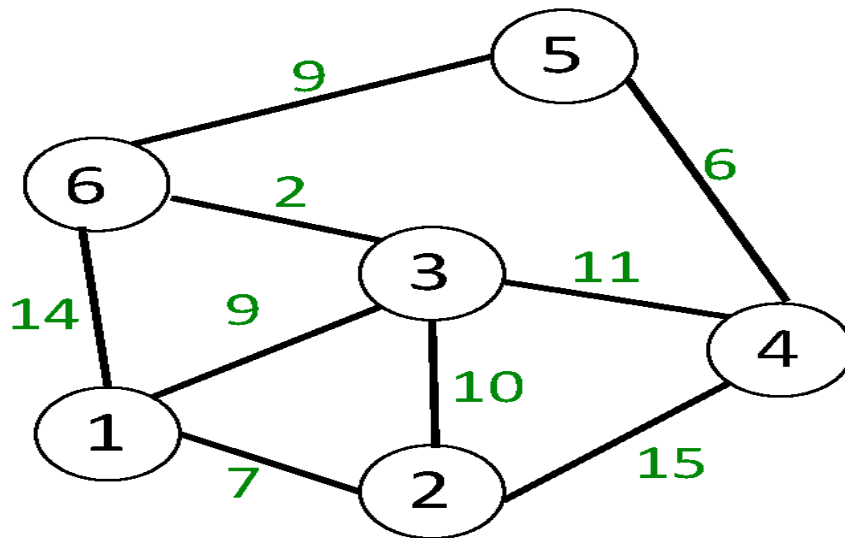
- **Background, Introduction and Application**
- **Implementation of Search in BST**
 - In CPP
 - In Java
- **Insertion in BST**
 - In CPP
 - In Java
- **Deletion in BST**
 - In CPP
 - In Java
- **Floor in BST**
 - In CPP
 - In Java
- **Self Balancing BST**
- **AVL Tree**
- **Red Black Tree**
- **Set in C++ STL**
- **Map in C++ STL**

HEAP

- **Introduction & Implementation**
- **Binary Heap**
 - Insertion
 - Heapify and Extract
 - Decrease Key, Delete and Build Heap

- **Heap Sort**
- **Priority Queue in C++**
- **PriorityQueue in Java**

GRAPH



- **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**
 - Implementation in CPP
 - Implementation in Java

- **Dijkstra's Shortest Path Algorithm**
 - Implementation in CPP
 - Implementation in Java
- **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**

REASON FOR CHOOSING DSA

If anyone preparing out for a tech interview with a product-based company or planning to do the same? almost every Big tech Faang company like Facebook, Google, Amazon, Apple, Microsoft, and many more. focuses more on the DSA skills of the candidates during the interviews. The reason behind it could be as DSA not only allows the interviewer to assess our programming or technical skills but also shows our problem-solving skills to come up with an optimized solution for the particular problem.

After that, I was having so many questions in my mind like how to start preparing for DSA to crack the interviews, where to get the quality learning resources, what should be the preparation strategy and many more? I look at so many resources out there on youtube but something is still missing and that was how to know whether my solution is correct or not and how can I modify or formate my solutions.

then I get to know about this course which helps to learn Data Structure and Algorithm concepts from basic to the advanced level. This self-paced course has been divided into **8 weeks** where I can learn the DSA from basic to advanced level and can practice questions & attempt the assessment tests.

In this course, I have also learned algorithmic techniques for solving various problems with full flexibility of time. Also, there is no need for any prior knowledge of Data Structure and Algorithms for this course - however, I was having a basic knowledge of C++ programming language, it was quite helpful.

Some of the prominent features of this course are :

- *Mastering DSA from basic to advanced level*
- *Solving problems which are asked in product-based companies*
- *Solve problems in contests similar to coding round for SDE role*
- *Premium video lectures by Sandeep Jain (Founder & CEO, GeeksforGeeks)*
- *Track-based learning & Assessment tests*
- *Quiz & Theory-based Learning*
- *Lifetime access to the Course*
- *Internship Opportunities at GeeksforGeeks*
- *Access to GeeksforGeeks Job Portal*

Also, I opted to **add on the doubt assistance facility** in this course that helps to improve my approach to solve a problem.

Contest solutions are available in Video mode in the contest section itself.

Furthermore, it helps me to prepare for interviews with top-notch companies like **Microsoft, Amazon, Adobe**, etc. by providing an extra edge to my problem-solving skill.

LEARNING OUTCOMES

A lot of beginners and experienced programmers avoid learning Data

Structures and Algorithms because it's complicated and they think that there is no use of all the above stuff in real life but there is a lot of implementation of DSA in daily life.

For example If we have to search our roll number in 2000 pages of Document how would we do that?

- If we try to search it randomly or in sequence it will take too much time.
- We can try another method in which we can directly go to page no. 1000 and we can see if our roll no. is there or not if not we can move ahead and by repeating this and eliminating we can search our roll no. in no time.

And this is called Binary Search Algorithm.

Two reasons to Learn Data Structure and Algorithms -

- If you want to crack the interviews and get into the product based companies
- If you love to solve the real-world complex problems.

I have learnt a vast number of topics like Trees, Graphs, Linked Lists, Arrays, etc. I understood their basics, their working, their implementation, and their practical use in the problems we face while we solve a problem using coding.

When we work in IT sector (Software or Programming part to be specific) we need to solve the problems and make programs write tons of code which will help us with the given problem and to write a program one needs to make different algorithms. Many algorithms combine to make a program. Now, algorithms are written in some languages but they are not dependent on them, one needs to make a plan and algorithm first then write it into any language whether it is C++ or JAVA or C or any other programming language. Algorithm is based on data structure and its implementation and working. So, basically one needs to have a good grip on DSA to work in programming sector.

When you ask someone to make a decision for something the good one will be able to tell you *"I chose to do X because it's better than A, B in these ways. I could have gone with C, but I felt this was a better choice because of this"*. 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 have to 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.

I learned about how to break a problem into pieces and then find the solution then how to make the desired algorithm which will help me to solve my respective problem.

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.

The completion of the summer training as well as the mini project have been absolutely beneficial to me in many respects.

Here, I will list out the major outcomes of the project that I have received.

- First and foremost, I got a chance to clearly understand the Data Structures available which will help to grasp the future concepts easily.
- I also got to know how to use space and time complexity efficiently in designing programs.
- After Learning the data structures properly, logic building also has become easier for me, making me more interested in coding.

Apart from the learning point of view, I believe the knowledge I have gained while completion of the summer training as well as the mini project will help me in coding interviews, as in various coding interviews, data structures are most important topic. Some other outcomes of the summer training and the mini project are:

- Better grasp over programming concepts.
- Better problem-solving skills.
- Good grasp over clean and efficient programming.
- Knowledge on time and space complexity.

- Learning new programming tricks and tips.
- Extended knowledge on C++.
- Insights on some of the commonly asked questions in the interview.
- Better preparation for the interviews.
- Getting to know about mathematical concepts used in programming and how to correlate the problem to mathematics.

The mini project has also enabled me to get a hands-on experience with my coding skills and got a chance to gauge myself. It enabled me to understand the practical application of what I learned in theory.

Project Legacy

The summer training had many technical and managerial lessons in store for me which I am glad that I explored.

Let us go through some of the major lessons that I got to learn while completing the training and the project.

- Better grasp on C++.
- Better grasp on Data Structures.
- Improved logic building.
- Better understanding of programming concepts.
- Knowledge of mathematical concepts and tips and tricks for efficient and fast programming.
- Space and time complexity analysis in programs.
- Time management for completion of several tasks.

Apart from that, I got to experience how a real world program comes to action after unification of several subprograms, functions, in this case.

Managerial skills such as how to maximize output within a specified time limit, how to devise strategies and planning the course of action could also be drawn as some of the most important outcomes of the mini project and the summer training.

Since, it has to be completed in a limited amount of time, time management skills were of great use and greatly improved too, as a result of which, better planning strategies came into scene and helped me to complete the report, the mini project and the training.

Taking about technical outcomes, they are in a huge number as they include everything from hands-on training to self-thinking of the solution of the problems.

The programming language, C++, combined with data structure provided me with a complete package that I can use in the future for my own benefit.

To sum it up, it has been very much pleasant experience for me throughout the completion of the project and the training as I got to learn new things and experience how the efficient coding is put into action.

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

CONCLUSION

I strongly believe that this course from geeks for geeks have helped me to create a strong basic foundation in data structures and algorithm. I have learnt many different data structures and algorithms in this and tried to implement in the problems given along with different topics. Although some problems were of a very hard level and I took help of the hints, but I had a better insight of the basic details of the data structures and algorithm. We can conclude that this course has been very helpful to make foundation in programming clearer. If I had not done this, I would have never been able to make such a good project.

1. Improved problem-solving skills to become a stronger developer.
2. Develop analytical skills on Data Structures and use them efficiently
3. Solve problems asked in product-based companies' interviews
4. Solve problems in contests like coding round for SDE role
5. Course Completion Certificate.

References

- [GeeksforGeeks | A computer science portal for geeks](#)
- [Biggest Online Tutorials Library \(tutorialspoint.com\)](#)
- [Programiz: Learn to Code for Free](#)
- [W3Schools Online Web Tutorials](#)

Thank You