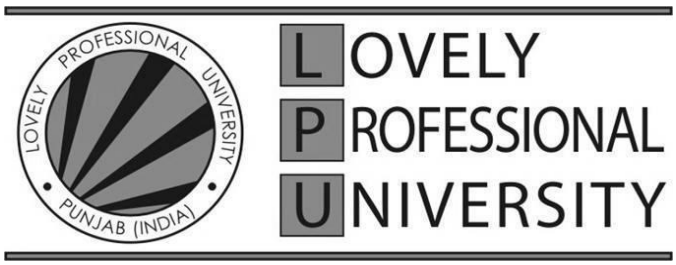

SIX WEEKS SUMMER TRAINING REPORT

On

Data Structures and Algorithms

Self-Paced Course

Submitted by

Name: Rudra Kaniya

Registration No. 11803187

Program Name:

BTech, Computer Science Engineering

School of Computer Science & Engineering

Lovely Professional University, Phagwara

(June-July, 2020)

COURSE CONTENT

1. Introduction
2. Mathematics
3. Bit Magic
4. Recursion
5. Arrays
6. Searching
7. Sorting
8. Matrix
9. Hashing
10. Strings
11. Linked List
12. Stack
13. Queue
14. Deque
15. Tree
16. Binary Search Tree
17. Heap
18. Graph
19. Greedy
20. Dynamic Programming

INTODUCTION

- **Analysis of Algorithm**
    - Background analysis through a Program and its functions. In computer science, the analysis of algorithms is the process of finding the computational complexity of algorithms – the amount of time, storage, or other resources needed to execute them.
- **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 (Ω Notation), Average (Big O)** and **Worst-case (Θ Notation)** explanation through a program. The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis.
- **Big O Notation**
- **Omega Notation**
- **Theta Notation**
- **Analysis of common loops**
    - Single, multiple and nested loops
- **Analysis of Recursion**
    - Various calculations through Recursion Tree method
- **Space Complexity**
    - The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of characteristics of the input. It is the memory required by an algorithm to execute a program and produce output.

MATHEMATICS

Count Digits The objective of this track is to familiarize the learners with the basics of Mathematics needed to solve some commonly asked and popular programming problems.

- **Finding the number of digits in a number.**
    - The task is to find the count of digits present in this number. A prime number is a whole number greater than 1, which is only divisible by 1 and itself
- **Quadratic Equations.**
- **Mean and Median.**
- **Prime Numbers.**
    - A prime number is a whole number greater than 1, which is only divisible by 1 and itself. LCM stands for Least Common Multiple. The lowest number which is exactly divisible by each of the given numbers is called the least common multiple of those numbers.
- **LCM and HCF**
    - LCM stands for Least Common Multiple. The lowest number which is exactly divisible by each of the given numbers is called the least common multiple of those numbers.
    - The term HCF stands for Highest Common Factor. The largest number that divides two or more numbers is the highest common factor (HCF) for those numbers.
- **Factorials**
    - In mathematics, the factorial of a number say N is denoted by N!. The factorial of a number is calculated by finding multiplication of all integers between 1 and N(both inclusive.)

- **Permutations and Combinations**
- **Modular Arithmetic**

BIT MAGIC

Objective: The objective of this track is to familiarize the learners with *Bitwise Algorithms* which can be used to solve problems efficiently and some interesting tips and tricks using Bit Algorithms.

- **Binary Representation:** We'll look at the binary representation of numbers.
- **Set and Unset:** We'll learn to set and unset the bits
- **Toggling:** We'll toggle the bits.
- **Bitwise Operators:** We'll use AND, OR, XOR, NOT, LShift, and RSHIFT operators.
- **Algorithms:** We'll get familiar with various bitwise algorithms that'll make problem solving easy.

- **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.

Operations with bits are used in Data compression (data is compressed by converting it from one representation to another, to reduce the space), Exclusive-Or Encryption (an algorithm to encrypt the data for safety issues). In order to encode, decode or compress files we have to extract the data at bit level. Bitwise Operations are faster and closer to the system and sometimes optimize the program to a good level.

RECURSION

Objective: The objective of this track is to familiarize the learners with the basics of *Recursion.*

- **Recursion Basics:** What exactly is recursion and why it's important.
- **Advantages**: Why is recursion code easier to write than it's the iterative counterpart.
- **Types:** We'll look at head recursion and Tail recursion.
- **Problems:** How do we approach a question that is to be solved by recursion.
- **Introduction to Recursion**
    - Operation of Right Shift and unsigned Right Shift. Recursion is a powerful algorithmic technique in which a function calls itself (either directly or indirectly) on a smaller problem of the same type in order to simplify the problem to a solvable state. Every recursive function must have at least two cases: the recursive case and the base case
- **Applications of Recursion**
    - Most computer programming languages support recursion by allowing a function to call itself from within its own code. Some functional programming languages do not define any looping constructs but rely solely on recursion to repeatedly call code.

- **Writing base cases in Recursion**
    - Factorial
    - N-th Fibonacci number

ARRAYS

Objective: The objective of this track is to familiarize the learners with *arrays.*

- **Arrays:** What are arrays and why are they used.
- **Basic Operations:** We'll look at basic array operations like insertion, deletion, and reversal.
- **Shifting and Rotation:** How to shift and rotate an array.
- **Sum Arrays:** How do we create a prefix sum array and when do we need it.
- **Sliding Window:** Using the sliding window to improve the complexity of our program.

- **Types of Arrays**
  - Fixed-sized array
  - Dynamic-sized array
- **Operations on Arrays**
  - Searching
  - Insertions
  - Deletion
  - Arrays vs other DS
  - Reversing - Explanation with complexity

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.

SEARCHING

Objective: The objective of this track is to familiarize the learners with *Searching Algorithms.*

- **Basics:** What exactly is searching and how it's helpful in the programming paradigm.
- **Types:** We'll look at Linear Search, Binary Search, and Ternary Search.
- **Implementation:** How to implement the search algorithms in CPP and Java.
- **Binary Search Iterative and Recursive**
- **Binary Search and various associated problems**
    - Index of First Occurrence in Sorted Array
    - Index of Last Occurrence in Sorted Array
    - Count of occurrences of x in sorted element
    - Count of 1s in a binary sorted array
    - Find an element in sorted and rotated array
    - Peak element
    - Find an element in an infinite sized sorted array
    - The square root of an integer
- **Two Pointer Approach Problems**
    - Find pair in an unsorted array which gives sum X
    - Find pair in a sorted array which gives sum X
    - Find triplet in an array which gives sum X

SORTING

Objective: The objective of this track is to familiarize the learners with *Sorting Algorithms.*

- **Basics:** What exactly is sorting and how it's helpful in the programming paradigm.
- **Basic Algorithms:** We'll look at bubble sort, insertion sort, selection sort.
- **Specialized algorithms:** Quick Sort, Merge Sort, Count Sort, Heap Sort.
- **Implementation:** How to use inbuilt CPP and Java sorting functions.
- **Implementation of C++ STL sort() function in Arrays and Vectors**
    - Time Complexities
- **Stability in Sorting Algorithms**
    - Examples of Stable and Unstable Algos
- **Insertion Sort**
- **Merge Sort**
- **Quick Sort**
    - Time and Space analysis
    - Choice of Pivot and Worst case
    - Tail call elimination
- **Cycle Sort**
- **Counting Sort**
- **Radix Sort**
- **Bucket Sort**
- **Overview of Sorting Algorithms**

MATRIX

Objective: The objective of this track is to familiarize the learners with *Matrices.*

- **Matrix:** The basic definition of the matrix.
- **Terminologies:** The various terminologies associated with a matrix.
- **Properties:** The most important properties of a matrix like commutative, associative, etc.
- **Implementation:** How we implement matrix in CPP and Java.
- **Operations:** How to add, subtract, and multiply matrices.
- **Rotation:** We'll learn the matrix rotation technique.
- **Introduction to Matrix**
    - A matrix is a rectangular array of numbers or symbols arranged in rows and columns. The C programs in this section perform the operations of Addition, Subtraction and Multiplication on the 2 matrices. The section also deals with evaluating the transpose of a given matrix.
- **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**

HASHING

Objective: The objective of this track is to familiarize the learners with *Hashing.*

- **Basics:** What exactly is the concept of hashing.
- **Types:** We'll look at Separate chaining and Open addressing.
- **Implementation:** How to use hashing in CPP and Java.
- **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

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used. Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements.

STRINGS

Objective: The objective of this track is to familiarize the learners with *Strings.*

- **Basics:** Brief introduction to strings.
- **Algorithms:** We'll look at various pattern matching algorithms in strings.
- **Implementation:** How to use strings in CPP and Java.
- **Discussion of String DS**
- **Strings in CPP**
    - C++ has in its definition a way to represent sequence of characters as an object of class. This class is called std:: string. String class stores the characters as a sequence of bytes with a functionality of allowing access to single byte character
    - std:: string stringName;
- A character array is simply an array of characters can terminate by a null character. A string is a class which defines objects that be represented as stream of characters.
- Operations on strings
    - Unordered Map Input Functions
    - **getline()** :- This function is used to store a stream of characters as entered by the user in the object memory.
    - **push_back()** :- This function is used to input a character at the end of the string.
    - **pop_back()** :- Introduced from C++11(for strings), this function is used to delete the last character from the string.

LINKED LIST

Objective: The objective of this track is to familiarize the learners with *Linked List.*

- **Basics:** Introduction to linked list with advantages and disadvantages.
- **Operations:** Inserting nodes and deleting them, traversing the list.
- **Implementation:** How to implement Linked list in CPP and Java.
- **Types:** We'll study singly linked lists, doubly linked lists, XOR linked lists, and circular linked lists.

Singly linked list is a type of data structure that is made up of nodes that are created using self-referential structures. Each of these nodes contain two parts, namely the data and the reference to the next list node. Only the reference to the first list node is required to access the whole linked list. This is known as the head. The last node in the list points to nothing so it stores NULL in that part.

- **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

Objective: The objective of this track is to familiarize the learners with *Stack.*

- **Basics:** Introduction to the stack data structure. We'll look at insertion and deletion order in a stack.
- **Operations:** Various operations like push, pop, top, empty.
- **Implementation:** How to implement Stack in CPP and Java using arrays and linked lists.
- **Expressions:** We'll study infix, postfix, prefix expressions and conversion from one to other. We'll take a look at the evaluation of these expressions.
- **Understanding the Stack data structure**
- **Applications of Stack**
- **Implementation of Stack in Array and Linked List**
    - In C++
- **Understanding getMin() in Stack with O(1)**

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.

QUEUE

Objective: The objective of this track is to familiarize the learners with *Queue.*

- **Basics:** Introduction to the queue data structure. We'll look at the insertion and deletion order in a queue.
- **Operations:** Various operations like enqueue, dequeue, front, rear.
- **Implementation:** How to implement a queue in CPP and Java using arrays and linked lists.
- **Types:** We'll look at the circular queue.
- **Stacks and Queues:** Implementing one using other.

Queues are a type of container adaptors which operate in a first in first out (FIFO) type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

- **Introduction and Application**
    - Queue is used when things don't have to be processed immediately, but have to be processed in First In First Out order like Breadth First Search

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

DEQUE

- **Introduction and Application**
  - Double ended queues are sequence containers with the feature of expansion and contraction on both the ends.
  - They are similar to vectors, but are more efficient in case of insertion and deletion of elements. Unlike vectors, contiguous storage allocation may not be guaranteed.
  - Double Ended Queues are basically an implementation of the data structure double ended queue. A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back. Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends.

- **Implementation**
  - In C++ STL
  - Specific libraries may implement deques in different ways, generally as some form of dynamic array. But in any case, they allow for the individual elements to be accessed directly through random access iterators, with storage handled automatically by expanding and contracting the container as needed.
  - Therefore, they provide a functionality similar to vectors, but with efficient insertion and deletion of elements also at the beginning of the sequence, and not only at its end. But, unlike vectors, deques are not guaranteed to store all its elements in contiguous storage locations: accessing elements in a deque by offsetting a pointer to another element causes undefined behavior.

TREE

Objective: The objective of this track is to familiarize the learners with *Trees.*

- **Basics:** Introduction to the tree data structure. How exactly is it represented?
- **Operations:** Creation, Insertion, Deletion and other operations like LCA.
- **Implementation:** How to implement Trees.
- **Traversals:** We'll look at inorder, preorder, postorder, level order and various other traversals.

A tree is a collection of nodes connected to each other by means of “edges” which are either directed or undirected. One of the nodes is designated as “Root node” and the remaining nodes are called child nodes or the leaf nodes of the root node. In general, each node can have as many children but only one parent node.

- **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

Objective: The objective of this track is to familiarize the learners with *BSTs.*

- **Basics:** Introduction to BSTs and properties.
- **Operations:** Creation, Insertion, Deletion and other operations like LCA.
- **Implementation:** How to implement BSTs.
- **Balancing:** How to balance a BST.
- **Background, Introduction and Application**
- **Implementation of Search in BST**
    - In CPP
- **Self Balancing BST**
- **AVL Tree**
- **Red Black Tree**
- **Set in C++ STL**
- **Map in C++ STL**
- **BST Introduction**

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

HEAP

Objective: The objective of this track is to familiarize the learners with *Heaps.*

- **Basics:** Introduction to Heap.
- **Operations:** Creation, Insertion, Deletion, Heapify.
- **Implementation:** How to implement Heaps and priority queues in CPP and Java.
- **Types:** How Min Heap and Max Heap are different.
- **Introduction & Implementation**
    - Heap data structure can be implemented in a range using STL which allows faster input into heap and retrieval of a number always results in the largest number i.e. largest number of the remaining numbers is popped out each time. Other numbers of the heap are arranged depending upon the implementation.

- **Binary Heap**
    - Insertion
    - Heapify and Extract
    - Decrease Key, Delete and Build Heap
- **Heap Sort**
    - Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end.
- **Priority Queue in C++**

GRAPH

Objective: The objective of this track is to familiarize the learners with *Graphs.*

- **Basics:** Introduction to Graphs. properties, and representations.
- **Traversals:** How to traverse the graphs using BFS and DFS.
- **Algorithms:** How to detect cycles, find shortest paths, find strongly connected components, etc. in a graph.

- **Introduction to 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.
- **Graph Representation**
  - Adjacency Matrix
  - Adjacency List in CPP and Java
  - Adjacency Matrix VS List
- **Breadth-First Search**
  - Applications
- **Depth First Search**
  - Applications

GREEDY

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to the global optimal solution are best fit for Greedy.

For example, consider the **Fractional Knapsack Problem**. The problem states that:

Given a list of elements with specific values and weights associated with them, the task is to fill a Knapsack of weight W using these elements such that the value of knapsack is maximum possible.

Objective: The objective of this track is to familiarize learners with *Greedy Algorithms.*

- **Basics:** Introduction to greedy algorithms.

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

DYNAMIC PROGRAMMING

Objective: The objective of this track is to familiarize the learners with *Dynamic Programming.*

- **Basics:** Introduction to Dynamic Programming, overlapping subproblems, and optimal substructures.
- **Approaches:** Top-down and bottom-up approaches to solving a DP problem.
- **Steps:** How to approach a DP problem.

- **Introduction**
  - Memoization Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.

- **Dynamic Programming**
  - Memoization
  - Tabulation

~ *END* ~