Annexure-I

DATA STRUCTURE AND ALGORITHMS -SELF PACED

GeeksforGeeks

A training report

Submitted in partial fulfillment of the requirements for the award of degree of

B.Tech(CSE)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

From 09/05/23 to 17/07/23

SUBMITTED BY

Name of student: Rohit Kumar

Registration Number: 12104675

Signature of the student: Rohit Kumar

TABLE OF CONTENTS

Annexure-II: Student Declaration

To whom so ever it may concern

I, **Rohit Kumar**, **12104675**, hereby declare that the work done by me on "**Data Structure And Algorithms – Self Paced**" from **May, 2023** to **July, 2023**, is a record of original work for the partial fulfillment of the requirements for the award of the degree, **B.Tech (CSE)**.

Name of the Student: Rohit Kumar (12104675)

Signature of the student: Rohit Kumar

Dated: 26/08/2023

Training Certificate from GeeksforGeeks

ACKNOWLEDGEMENT

Embarking on the intricate journey of mastering Data Structures and Algorithms - Self Paced during my time at Lovely Professional University, I, Rohit Kumar, am compelled by the sincerest sense of gratitude to acknowledge those who have gracefully paved my path through this realm of algorithms, data structures, and logical problem-solving. To the compilers that silently translated my raw code into executable marvels, and to the debuggers that patiently untangled the knots of confusion, your contribution has been invaluable. GeeksforGeeks, your community's collective wisdom has been a guiding compass through syntax intricacies and conceptual enigmas. The virtual environment of coding challenges turned into a classroom, where fellow learners studying Data Structures and Algorithms - Self Paced became companions in the pursuit of knowledge. Each problem solved was a stepping stone, each concept grasped, a building block. A nod of appreciation to the cups of inspiration filled with caffeine, keeping my creativity awake during countless coding sessions. The keyboard beneath my fingers transformed into an orchestra, playing the symphony of algorithms with finesse. Amidst the keystrokes, a tip of the hat to the humble instant noodles, the unsung heroes fueling my late-night coding endeavours. But this journey isn't mine alone to traverse. The constellations of knowledge from GeeksforGeeks, especially through my course on Data Structures and Algorithms - Self Paced at Lovely Professional University, guided me through the dense fog of errors and illuminated the way through the maze of optimization. To my fellow learners of Data Structures and Algorithms - Self Paced, who turned competition into collaboration, I extend my gratitude for the camaraderie that makes this journey meaningful beyond the lines of code. As I navigate the landscape woven with 1s and 0s, I extend my heartfelt appreciation for the seamless guidance, the harmonious lessons, and the elegant solutions that have shaped my path in Data Structures and Algorithms - Self Paced.

With sincere appreciation,

Rohit Kumar

Chapter-1 INTRODUCTION

The Data Structures and Algorithms - Self Paced course through GeeksforGeeks is a comprehensive package that helped me learn Data Structures, Algorithms, and various problem domains from Beginner to Advanced levels. The course curriculum has been divided into 2 Months, during which I engaged with numerous practice questions and assessment tests. The course offers a wealth of programming challenges that helped me learn all about DSA, Number Theory, and Dynamic Programming. It facilitated the development of algorithms, problem-solving techniques, and the underlying logic.

This course includes video recordings and weekly live problem-solving and analysis sessions, promoting interactive learning. The content of this course will be available to me for one year once I am enrolled. This flexible schedule allowed me to learn various algorithms for solving diverse problems without time constraints.

While some prior knowledge of Data Structures and Algorithms is beneficial, a basic understanding of programming languages like C++ or Java is recommended. In today's competitive landscape, proficiency in Data Structures and Algorithms - Self Paced is vital for placement in any company Data Structure and Algorithms is prerequisites for resolving a variety of problems, particularly in product-based companies. Engaging with this course significantly enhanced my problem-solving skills.

For more interaction and details about the Data Structures and Algorithms - Self Paced course, I have been actively participating in discussions and seeking clarification on GeeksforGeeks platforms. These platforms have become a repository of invaluable insights, connecting me with a community of learners who share a common pursuit of mastering this intricate discipline. The opportunity to engage in meaningful discussions, seek guidance, and exchange ideas has further enriched my learning experience.

As I reflect on my journey through the Data Structures and Algorithms - Self Paced course, I am filled with gratitude for the comprehensive education, the interactive learning experiences, and the unwavering support of the GeeksforGeeks community. The skills I have acquired extend far beyond the realm of coding; they have equipped me with a holistic problem-solving approach that will undoubtedly shape my career and personal growth.

Chapter-2 TECHNOLOGY LEARNT

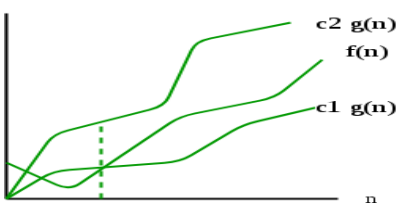
I opted Data Structure And Algorithm -Self-Paced. It had 11 units which was further divided into chapter and then topics so during my whole eight weeks I learned the following:

- **ANALYSIS OF ALGORITHMS**
  - **Analysis of Algorithms**
    - ✓ In this I learned about background analysis through a Program and its functions.
    - ✓ Studied program analysis and function behaviors for performance evaluation.
  - **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**
    - ✓ Learned about best, Average and Worst case explanation through a program.
  - **Θ Notation**

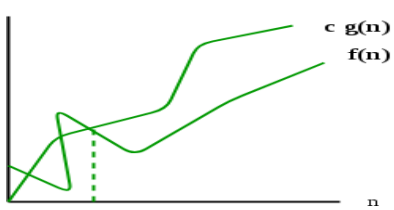
    Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}
  - **Big O Notation**

    O(g(n)) = {f(n): there exist positive constants c and n0 such that 0 <= f(n) <= c*g(n) for all n >= n0}
  - **Ω Notation**

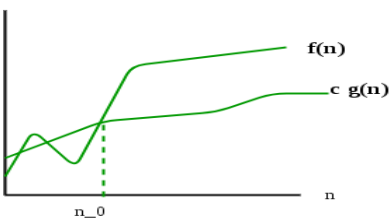
    Ω (g(n)) = {f(n): there exist positive constants c and n0 such that 0 <= c*g(n) <= f(n) for all n >= n0}



f(n) = theta(g(n))



f(n) = O(g(n))



f(n) = Omega(g(n))

- **Space Complexity**
  - ✓ Explored the memory used by algorithms during execution.
  - ✓ Analyzed efficiency in managing memory resources.
  - ✓ Grasped the impact of memory usage on overall performance.
- **Auxiliary Space**
  - ✓ Investigated extra memory usage apart from input data by algorithms.
  - ✓ Investigated algorithms' memory footprint.
  - ✓ Understood the significance of managing auxiliary space for efficient algorithm execution.

- **Analysis of Common loops**
  - ✓ Studied different loop structures, including Simple, multiple and nested loops
  - ✓ Analyzed their impact on program efficiency and gained insights into loop optimization.

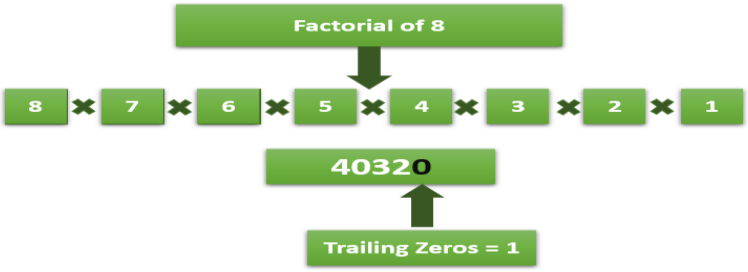
➢ **MATHEMATICS**

Explored various mathematical operations and concepts. Understood how mathematical principles underpin various problem-solving approaches.

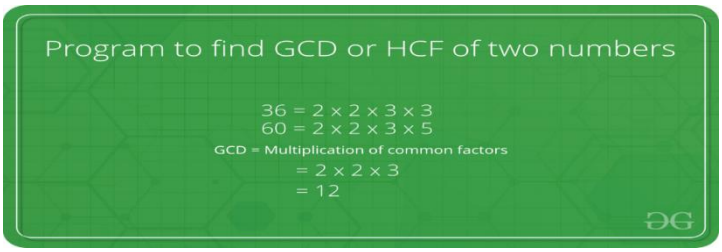
- **Count Digits**
- ✓ Learned to count the number of digits in a given number.
- ✓ Mastered the technique of counting digits in a number, a fundamental skill in data manipulation and algorithm design**.**
- **Palindrome Numbers**
  - ✓ Explored identifying numbers that read the same forwards and backwards.
  - ✓ Explored the identification of numbers that read the same forwards and backwards, understanding symmetry in numbers.
- **Factorial of a Number**
  - ✓ Studied the product of all positive integers up to a given number.

- ✓ Computed the product of all positive integers up to a given number, grasping its significance in permutations, combinations, and series.

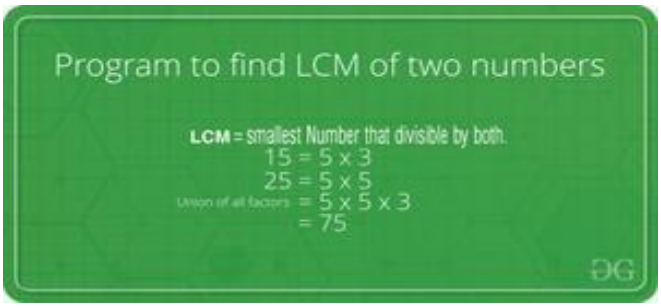
- **Trailing Zeros in Factorial**
  - ✓ Learned methods to determine the count of trailing zeros in factorials, essential for calculating the powers of prime factors.



- **GCD or HCF of two Numbers**
  - ✓ Delved into finding the greatest common divisor or highest common factor of two numbers, crucial in simplifying fractions and number analysis.

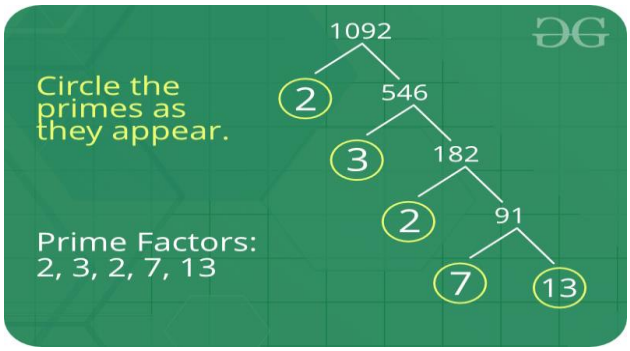


- **LCM of Two Numbers**
  - ✓ Mastered calculating the least common multiple of two numbers
  - ✓ A vital concept in diverse mathematical contexts such as arithmetic and algebra.



- **Check for Prime**
  - ✓ Explored techniques to verify the primality of a number,
  - ✓ A fundamental skill in number theory and cryptography.

- **Prime Factors**
  - ✓ Studied prime factors, the building blocks of numbers, and their relevance in various mathematical and algorithmic scenarios.

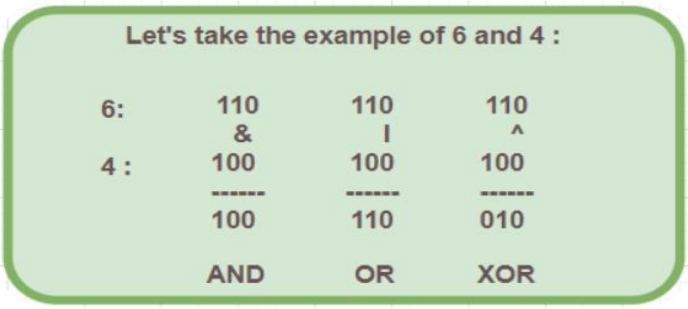


- **All Divisors of a Number**
  - ✓ Investigated divisors that evenly divide a given number,
  - ✓ Gaining insights into number properties and relationships.
- **Sieve of Eratosthenes**
  - ✓ Mastered an efficient algorithm to find prime numbers within a given range,
  - ✓ A foundational technique in number theory and cryptography.
- **Computing Power**
  - ✓ Explored algorithms for efficient exponentiation,
  - ✓ Enabling faster calculations of powers in various computational tasks.
- **Modular Arithmetic**
  - ✓ Studied the arithmetic of remainders,
  - ✓ A crucial tool in cryptography, number theory, and various computing applications.
- **Iterative Power**
  - ✓ Learned techniques for iteratively calculating exponentiation,

- ✓ Offering an alternative approach to obtaining powers efficiently.

- **BIT MAGIC**
  - **Bitwise Operators in CPP**

    Explored the usage of bitwise AND, OR, XOR, shifts, and complements in C++, acquiring low-level data manipulation skills for optimization.



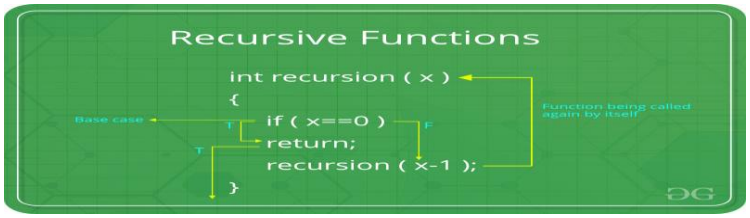
    - ✓ Binary AND - &
    - ✓ Binary OR - |
    - ✓ Binary XOR - ^
    - ✓ Left Shift - <<
    - ✓ Right Shift - >>
    - ✓ One's Complement - ~
  - **Bitwise Operator in Java**
    - ✓ Mastered bitwise AND, OR, XOR, shifts, and complements in Java, enabling efficient data manipulation and optimization**.**
    - ✓ AND, OR, XOR, Left Shift, Right, Shift, One's Complement
  - **Binary Representation of Negative Numbers**
    - ✓ Understood how negative integers are represented using two's complement in binary,
    - ✓ A fundamental concept in computer arithmetic and data storage.
  - **Check if Kth bit is set or not**
    - ✓ Explored techniques to determine if the Kth bit of a number is set (1) or not (0) using bitwise operations**.**
  - **Count Set Bits**

- ✓ Mastered the skill of counting the number of set (1) bits in a binary representation of a number,
- ✓ Essential for various algorithms.

- **Power of Two**
  - ✓ Studied methods to determine whether a number is a power of two, using bitwise operations to efficiently check the bit patterns.
- **One Odd Occurring**
  - ✓ Explored an efficient algorithm to find the element occurring an odd number of times in an array where all other elements occur even times.
- **Two Odd Occurring**
  - ✓ Delved into finding two elements occurring an odd number of times in an array, using bitwise XOR operations.
- **Power Set using Bitwise**
  - ✓ Learned the technique to generate the power set of a set using bitwise operations, a fundamental concept in combinatorics and algorithms.

➢ RECURSION

- **Recursion Introduction**
  - ✓ Explored the concept of recursive function calls and its use in solving problems by breaking them down into simpler sub-problems.



- **Applications of Recursion**
  - ✓ Studied various practical applications of recursion, including factorial calculation,
  - ✓ Fibonacci sequence generation, and more.
- **Recursion Output Practice**
  - ✓ Gained hands-on experience with recursion by practicing solving problems and tracing recursive function calls for output.

- **Print N to 1 Using Recursion**
  - ✓ Mastered printing numbers from N to 1 using recursive function calls, understanding the process of function stack and unwinding**.**
- **Print 1 to N Using Recursion**
  - ✓ Explored printing numbers from 1 to N using recursive function calls,
  - ✓ Emphasizing the sequence of function calls and their execution.
- **Tail Recursion**
  - ✓ Learned about tail recursion, a type of recursion where the recursive call is the last operation in the function.

```
// This is a Tail Recursion

void printN(int N)
{
    if(N==0)
        return;
    else
        cout<<N<<" ";

    printN(N-1);
}
```

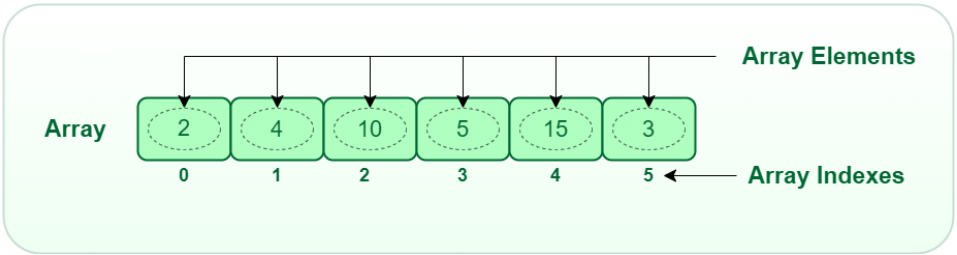
- **Natural Number Sum using Recursion**
  - ✓ Studied the recursive approach to calculate the sum of natural numbers up to a given limit,
  - ✓ Grasping the concept of base case and recursive step.
- **Palindrome Check using Recursion**
  - ✓ Explored using recursion to check if a given string is a palindrome,
  - ✓ Deepening the understanding of breaking problems into smaller instances.
- **Sum of Digits Using Recursion**
  - ✓ Learned the recursive approach to calculate the sum of digits of a number,
  - ✓ Practicing breaking down problems into simpler sub-problems**.**
- **Rope Cutting Problem**
  - ✓ Explored solving the rope-cutting problem using recursion,
  - ✓ Understanding the breakdown of a complex problem into smaller instances.
- **Generate Subsets**

- ✓ Mastered generating all possible subsets of a set using recursion, an essential concept in combinatorics and algorithms.

- **Tower of Hanoi**
  - ✓ Studied solving the classic Tower of Hanoi problem using recursion,
  - ✓ Gaining insights into recursive problem-solving strategies.
- **Josephus Problem**
  - ✓ Explored solving the Josephus problem using recursion,
  - ✓ Understanding how recursive patterns can be used for unique problem scenarios.
- **Subset Sum Problem**
  - ✓ Learned about solving the subset sum problem using recursion,
  - ✓ A fundamental concept in dynamic programming and combinatorial optimization.
- **Printing all Permutations**
  - ✓ Studied generating all permutations of a given string using recursion,
  - ✓ Understanding the permutation concept and backtracking.

➢ ARRAYS

- **Introduction to Arrays**
  - ✓ Explored the basic concept of arrays as data structures,
  - ✓ Understanding how elements are stored in contiguous memory locations.



- **Array Types**
  - ✓ Learned about various array types, including one-dimensional
  - ✓ Multi-dimensional, and jagged arrays, and their applications.
- **Vector in C++**

- ✓ Explored the dynamic array-like container "vector" in C++,
- ✓ Understanding its resizable nature and various methods for data manipulation.

- **ArrayList in Java**
  - ✓ Mastered the ArrayList class in Java, a dynamically resizing array implementation with methods for efficient data management**.**
- **Operations on Arrays**
  - ✓ Studied common operations like insertion, deletion, searching, and sorting on arrays,
  - ✓ Gaining a foundational understanding of array manipulation.
- **Reverse an Array**
  - ✓ Explored the algorithm to reverse the elements of an array,
  - ✓ Understanding the swapping process for efficient reversal.
- **Frequencies in a Sorted Array**
  - ✓ Learned techniques to count the frequency of elements in a sorted array,
  - ✓ Understanding the efficient counting process.
- **Maximum subarray sum**
  - ✓ Studied algorithms to find the maximum sum of a subarray within a given array,
  - ✓ Gaining insights into dynamic programming and greedy approaches.
- **Sliding Window Technique**
  - ✓ Explored the sliding window technique for efficient subarray and substring-related problems,
  - ✓ Understanding how to optimize using a window approach**.**
- **Prefix Sum**
  - ✓ Learned about the prefix sum array and its applications in solving problems like range sum queries and subarray sum calculations.
- **Equilibrium Point**
  - ✓ Studied finding the equilibrium point in an array, where the sum of elements on both sides is equal,
  - ✓ Understanding the approach to reach a solution**.**

- **SEARCHING**

- **Binary Search**
  - ✓ Mastered the binary search algorithm for efficient searching in sorted arrays, learning the divide-and-conquer strategy**.**

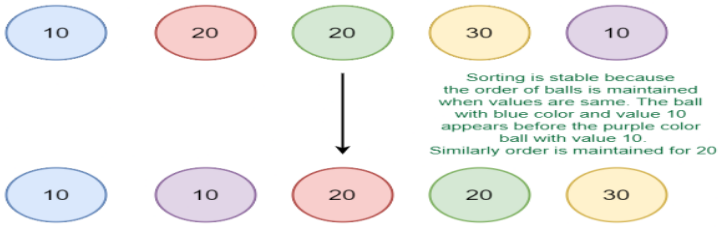
- **Binary Search Functions in C++ STL**
  - ✓ Explored the built-in binary search functions provided by the C++ Standard Template Library (STL) for efficient searching in containers.
- **BinarySearch using Built-in Methods in Java**
  - ✓ Learned to perform binary search using built-in methods like Arrays.binarySearch() in Java,
  - ✓ Enhancing searching efficiency.
- **Ternary Search**
  - ✓ Studied the ternary search algorithm for finding the maximum or minimum value in a unimodal function,
  - ✓ Understanding the trisection approach.
- **Square root**
  - ✓ Explored algorithms to calculate the square root of a number with efficient methods like the binary search approach.
- **Find a Peak Element**
  - ✓ Learned about finding a peak element in an array, understanding the application of binary search to locate local maxima.

➢ SORTING

- **Introduction to Sorting**
  - ✓ Gained an overview of sorting algorithms, understanding their significance in data organization and search optimization.
- **Sort in C++ STL**

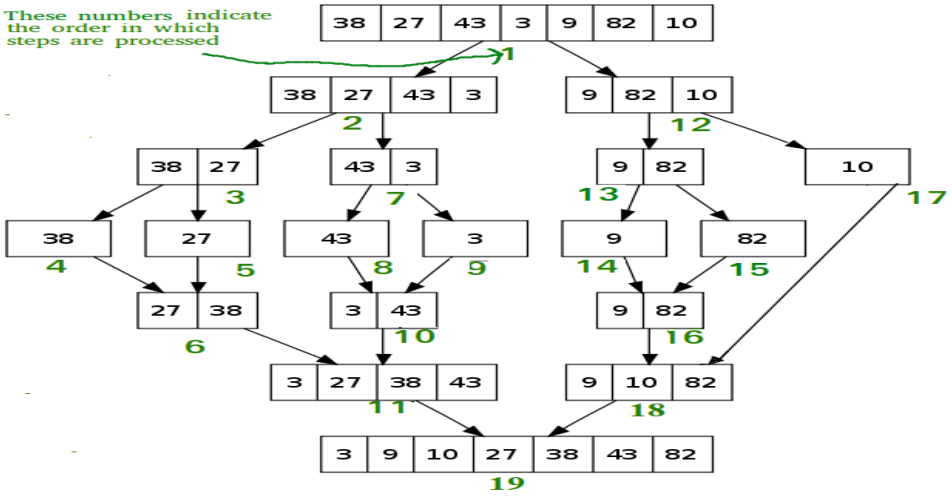
- ✓ Explored sorting algorithms provided by the C++ STL, such as std::sort(), for efficient array sorting.

- **Sorting using Built-in methods in Java**
  - ✓ Learned about array sorting using built-in methods like Arrays.sort() in Java,
  - ✓ Understanding how to leverage Java's libraries.
- **Arrays.sort() in Java**
  - ✓ Mastered Java's built-in sorting method, Arrays.sort(), understanding how to efficiently sort arrays and lists.
- **Collections.sort() in Java**
  - ✓ Studied the Collections.sort() method in Java, which provides sorting for lists and collections,
  - ✓ Enhancing data organization**.**
- **Stability in Sorting Algorithm**
  - ✓ Explored the concept of stability in sorting algorithms, understanding how it impacts the relative order of equal elements**.**



- **Types of Sorting:**
  - ✓ Bubble Sort
  - ✓ Selection Sort
  - ✓ Insertion Sort
  - ✓ Merge Sort –

    : Studied the Merge Sort algorithm, understanding the divide-and-conquer strategy to achieve efficient sorting.



- ✓ Quick Sort

➢ MATRIX

o Introduction to Matrix

- ✓ Understood the concept of a matrix as a two-dimensional data structure.
- ✓ Explored how it's used to represent data in rows and columns.

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

o 2D Vector in C++

- ✓ Learned to implement a matrix using 2D vectors in C++, a flexible way to manage 2D arrays dynamically.

o Implementing Matrix using 2D Arrays in Java

- ✓ Explored the implementation of matrices using 2D arrays in Java, understanding how to manipulate elements efficiently.

o Matrix Operations (Addition, Subtraction, Multiplication)

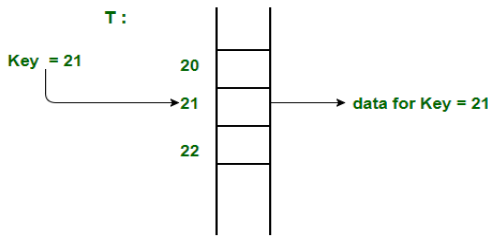
- ✓ Studied fundamental operations on matrices,
- ✓ Including addition, subtraction, and multiplication, to perform arithmetic calculations.

- **Transpose of a Matrix**
  - ✓ Learned to compute the transpose of a matrix, exchanging rows and columns to create a new matrix.
- **Matrix Rotation**
  - ✓ Explored rotating a matrix by 90 degrees,
  - ✓ Understanding the manipulation of elements to achieve the rotation effect**.**
- **Spiral Traversal of Matrix**
  - ✓ Mastered the technique of traversing a matrix in a spiral order,
  - ✓ Gaining insights into efficiently navigating through the matrix elements**.**
- **Determinant of a Matrix | Explanation**
  - ✓ Studied how to calculate the determinant of a square matrix, a key concept in linear algebra with applications in various fields.

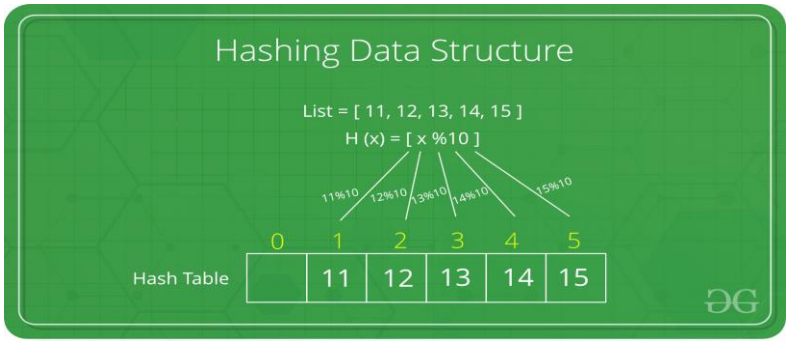
➢ **HASHING**

- **Introduction to Hashing**
  - ✓ Understood the concept of hashing, where data is stored and retrieved using hash functions, enhancing data access efficiency.
- **Hashing Application**
  - ✓ Explored practical applications of hashing, such as implementing databases,
  - ✓ Handling collisions, and ensuring data integrity.

- **Direct Address Table**
  - ✓ Learned the concept of direct addressing, a basic form of hashing, where each key maps to a specific index in an array.



- **Hashing Functions**
  - ✓ Studied hash functions, which convert data into fixed-size values, critical for efficient data retrieval and storage



- **Collision Handling**
  - ✓ Explored techniques to manage collisions, instances where multiple keys map to the same hash value,
  - ✓ Ensuring accurate data retrieval.
- **Implementation of Chaining**
  - ✓ Mastered the technique of chaining to handle collisions, where each hash table index contains a linked list of elements.
- **Open Addressing**
  - ✓ Studied open addressing, an alternative collision resolution method, where colliding elements are placed in nearby locations.
- **Double Hashing**
  - ✓ Explored double hashing as a collision resolution strategy,
  - ✓ Understanding how it involves using two hash functions.
- **Chaining vs Open Addressing**
  - ✓ Comparatively studied chaining and open addressing collision resolution methods, understanding their pros and cons.
- **unordered_set in C++ STL**
  - ✓ Learned about the unordered_set container in C++ STL, a hash table-based data structure for storing unique elements.
- **unordered_map in C++ STL**

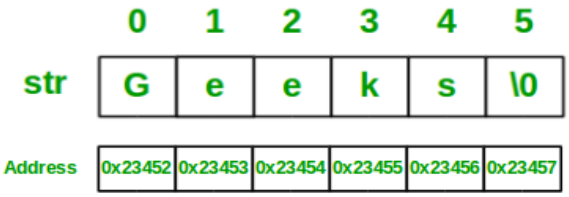
- ✓ Explored the unordered_map container in C++ STL, used to store key-value pairs with fast data retrieval based on hashing

- **HashSet in Java**
  - ✓ Mastered the HashSet class in Java, a hash table-based data structure for storing unique elements,
  - ✓ Essential for quick data access.
- **HashMap in Java**
  - ✓ Studied the HashMap class in Java, used for key-value pair storage and fast data retrieval through hashing.
- **Longest Consecutive Subsequence**
  - ✓ Explored an algorithmic problem of finding the longest consecutive subsequence in an array,
  - ✓ Applying hash-based techniques.

➢ STRINGS

- **Introduction to Strings**

  Understood the concept of strings as sequences of characters, exploring their importance in programming and text manipulation.

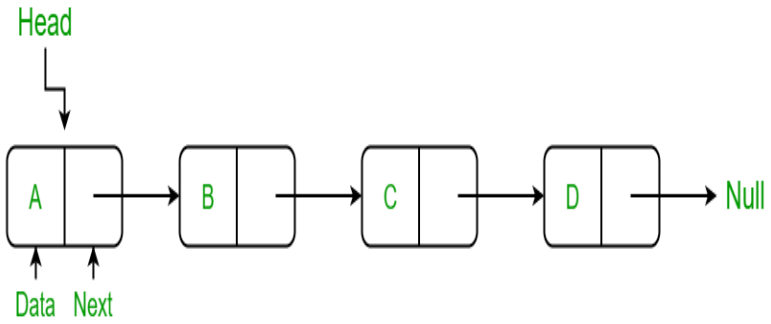


- **Strings in C++**
  - ✓ Learned about string manipulation in C++, including concatenation, substring extraction, and other useful operations.
- **Strings in Java**
  - ✓ Explored Java's String class, understanding its methods for character manipulation, substring extraction, and more.
- **Naive Pattern Searching Algorithm**
  - ✓ Studied the basic pattern searching algorithm that checks for the occurrence of a given pattern in a text using a sliding window.

- **Rabin-Karp Algorithm for Pattern Searching**
  - ✓ Mastered the Rabin-Karp algorithm for pattern searching, which uses hashing to quickly find patterns in a text.
- **KMP Algorithm for Pattern Searching**
  - ✓ Explored the Knuth-Morris-Pratt algorithm for pattern searching,
  - ✓ Emphasizing efficient text searching through pattern preprocessing.

➢ LINKED LIST

- **Introduction**
  - ✓ Understood the concept of linked lists as dynamic data structures, grasping the notion of nodes interconnected via pointers.



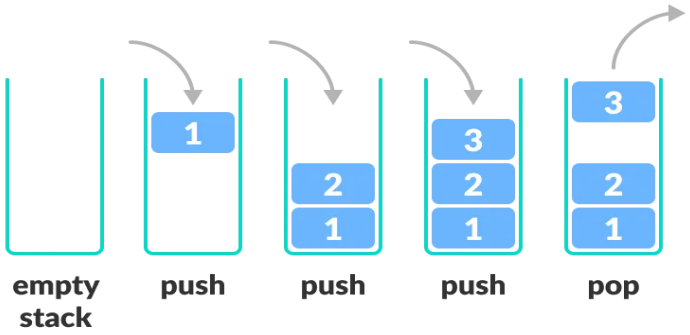
- **Singly Linked List**
  - ✓ Learned the implementation and manipulation of singly linked lists, where each node points to the next node in the sequence.
- **Doubly Linked List**
  - ✓ Explored doubly linked lists, where each node points both to the next and previous nodes,
  - ✓ Enhancing navigation flexibility.
- **Circular Linked List**
  - ✓ Studied circular linked lists, where the last node points back to the first,
  - ✓ Creating a circular structure.
- **Circular Doubly Linked List**

- ✓ Mastered the implementation and manipulation of circular doubly linked lists, Combining features of both circular and doubly linked lists.

➢ STACK

○ Introduction

- ✓ Understood the stack data structure and its LIFO (Last In, First Out) behavior,
- ✓ Exploring its applications in various programming scenarios.



○ Applications of Stack

- ✓ Explored the practical applications of stacks in tasks like function calls, expression evaluation, and solving algorithmic problems.

○ Stack in C++ STL

- ✓ Learned about the stack container in C++ STL, which provides methods for push, pop, and other operations essential for stack management.

○ Stack in Java Collection

- ✓ Explored Java's Stack class, understanding its methods for push, pop, and peek,
- ✓ Useful in managing function calls and more.

○ Infix to Postfix

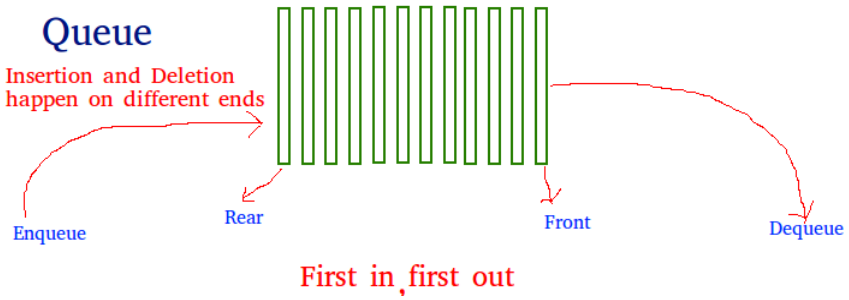
- ✓ Mastered the conversion of infix expressions to postfix using stacks, an essential step in evaluating mathematical expressions.

- **Infix To Prefix**
  - ✓ Studied the conversion of infix expressions to prefix (Polish) notation using stacks, useful in expression evaluation.

➢ QUEUE

- **Introduction**

  Understood the queue data structure and its FIFO (First In, First Out) behavior, exploring its applications in data processing.



- **Implementing Queue in C++ and Java using Built-in Classes**
  - ✓ Learned to implement queues using built-in classes in C++ and Java, gaining insights into enqueueing and dequeueing operations
- **Implementing Stack using Queue**
  - ✓ Explored how to implement a stack using queues, understanding how two queues
- **Implementing Queue using Stack**
  - ✓ Studied implementing a queue using stacks, using two stacks to mimic queue operations for efficient data management.

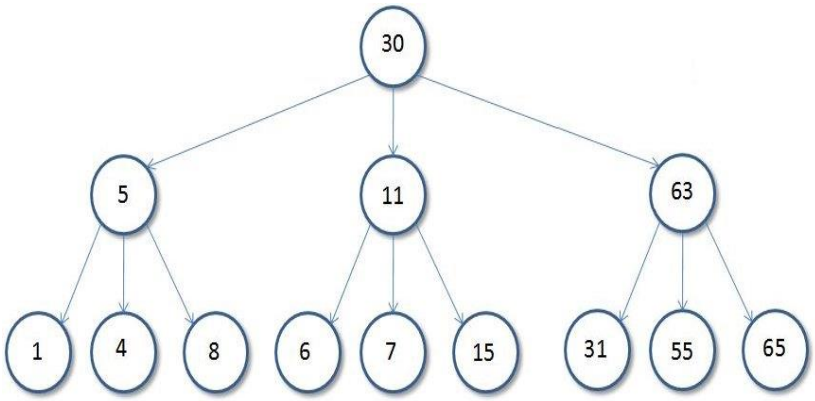
➢ DEQUEUE

- **Introduction**
  - ✓ Understood the concept of a double-ended queue (deque), a versatile data structure that supports insertion and deletion from both ends.
- **Deque in C++ STL**
  - ✓ Explored the deque container in C++ STL, learning about its dynamic resizing and insertion/deletion features from both ends.

- **Deque in Java**
  - ✓ Mastered the Deque interface in Java, understanding its implementation classes ArrayDeque and LinkedList for flexible data management.

➢ TREE

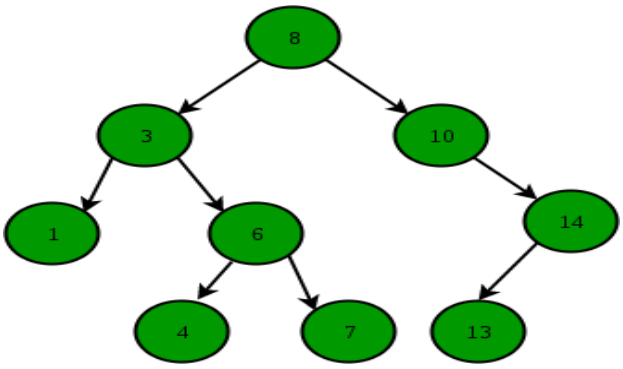
- **Introduction**
  - ✓ Understood the concept of trees as hierarchical data structures, exploring their applications in various programming scenarios**.**



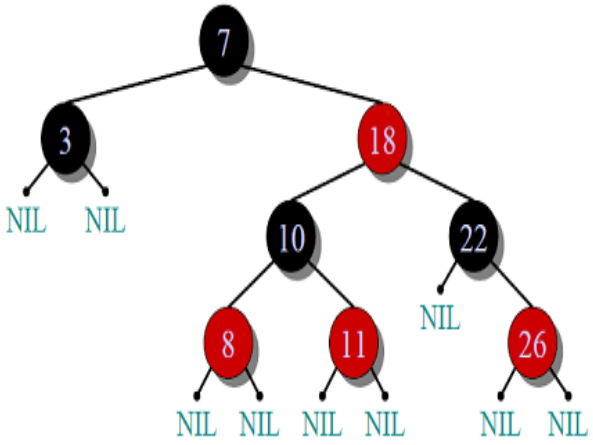
- **Application of Tree**
  - ✓ Explored the applications of trees in data representation, file systems, hierarchical structures, and more.
- **Binary Tree**
  - ✓ Learned about binary trees, where each node has at most two children, understanding their traversal and manipulation.
- **Binary Tree Traversals**
  - ✓ Studied the algorithms for in-order, pre-order, and post-order traversals of binary trees, understanding their sequencing.
- **Implementation of Inorder, Preorder and Postorder Traversal**
  - ✓ Mastered implementing and understanding in-order, pre-order, and post-order traversals through recursive and iterative approaches.

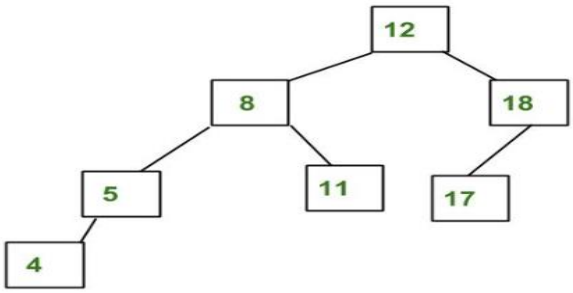
➢ BINARY SEARCH TREE

- **Introduction**
  - ✓ Understood the concept of binary search trees (BSTs), where each node's value is greater than values in its left subtree and smaller than those in its right subtree.

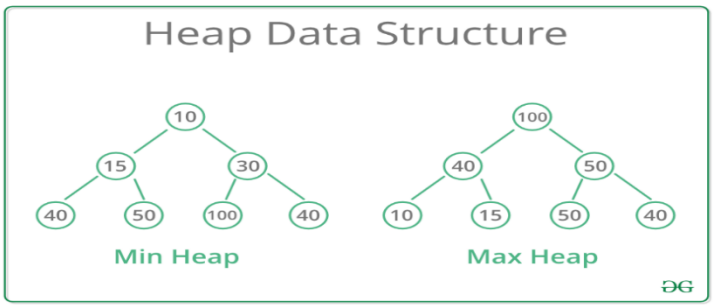


- **Floor in BST**
  - ✓ Learned to find the floor value of a given key in a binary search tree, the largest value less than or equal to the given key.
- **Ceil in BST**
  - ✓ Explored finding the ceiling value of a given key in a binary search tree, the smallest value greater than or equal to the given key.
- **Introduction to AVL Tree**
  - ✓ Studied AVL trees, a self-balancing binary search tree where the heights of the two child subtrees differ by at most one**.**
- **Red Black Tree**
  - ✓ Learned about red-black trees, a type of self-balancing binary search tree with constraints on coloring and rotations.
- **Applications of BST**
  - ✓ Explored practical applications of binary search trees in scenarios like searching, insertion, and deletion operations.
- **Self-Balancing Tree**
  - ✓ Studied self-balancing trees like AVL and red-black trees, which maintain their balance during insertion and deletion operations.





- **HEAP**
  - **Binary Heap Introduction**
    - ✓ Understood binary heaps, a complete binary tree where each parent node is greater (or smaller) than its children.



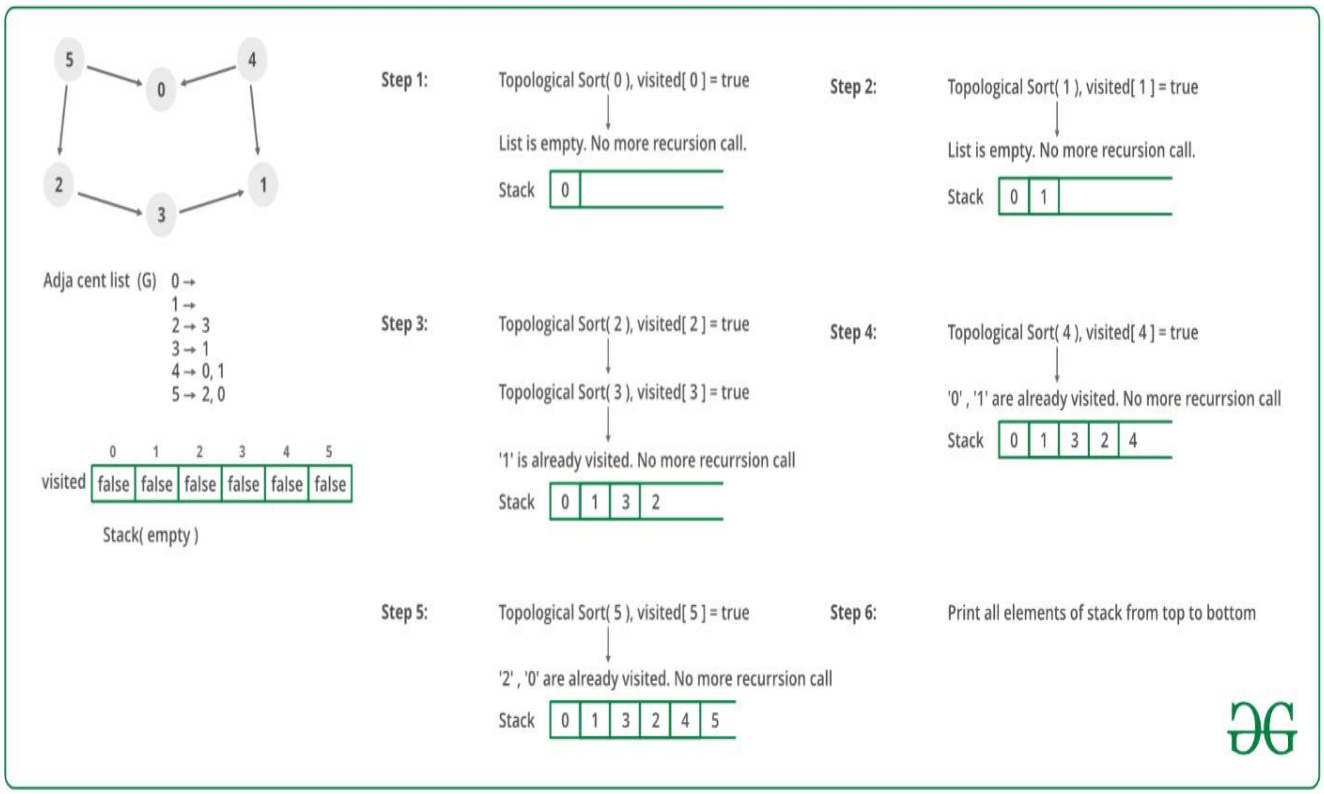
  - **Insertion and Deletion in Heap**
    - ✓ Mastered the insertion and deletion operations in binary heaps, ensuring heap properties are maintained.
  - **Operations of Heap Data Structure**

- ✓ Explored various heap operations like building a heap, extracting the root element, and heapifying to maintain heap properties**.**

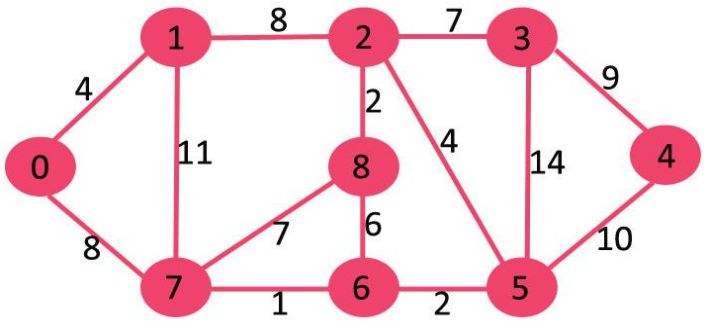
- **Heap Sort**
  - ✓ Studied the Heap Sort algorithm, which involves building a heap and repeatedly extracting the root element to achieve sorted order.

➢ GRAPH

- **Introduction**
  - ✓ Understood graphs as a collection of nodes (vertices) connected by edges, exploring their applications in modeling relationships.
- **Graph Representation**
  - ✓ Explored various graph representations including adjacency matrix and adjacency list, understanding their trade-offs and applications**.**
- **Adjacency Matrix and List Comparison**
  - ✓ Comparatively studied the adjacency matrix and adjacency list representations for graphs, understanding their strengths and weaknesses.
- **Breadth First Traversal of a Graph**
  - ✓ Mastered the Breadth First Search (BFS) algorithm for traversing a graph, understanding its level-wise exploration strategy.
- **Applications of BFS**
  - ✓ Explored practical applications of Breadth First Search in finding shortest paths, network analysis, and more.
- **Depth First Traversal of a Graph**
  - ✓ Studied the Depth First Search (DFS) algorithm for graph traversal, grasping its depth-first exploration strategy.
- **Topological Sorting**
  - ✓ Learned about topological sorting, an algorithm to linearly order the vertices of a directed acyclic graph based on dependencies.



- **Dijkstra's Algorithm**
  - ✓ Explored Dijkstra's algorithm for finding the shortest path in a weighted graph, understanding its greedy approach.

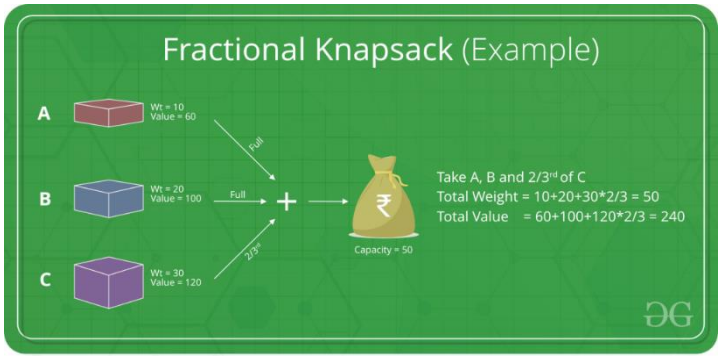


➢ GREEDY

- **Introduction**

- ✓ Understood the greedy algorithm paradigm, where decisions are made at each step to maximize immediate benefits without considering future consequences

- **Activity Selection Problem**
  - ✓ Studied the Activity Selection problem, a real-world scenario where tasks with start and finish times need to be selected optimally**.**
- **Fractional Knapsack Problem**
  - ✓ Learned about the Fractional Knapsack problem, where items with weights and values need to be chosen to maximize value within a weight limit.



- **Job Sequencing Problem**
  - ✓ Explored the Job Sequencing problem, where jobs with deadlines and profits need to be scheduled to maximize profit.
- **Huffman Coding**
  - ✓ Studied Huffman coding, a technique to compress data by assigning shorter codes to more frequent characters, optimizing space usage.

➢ BACKTRACKING

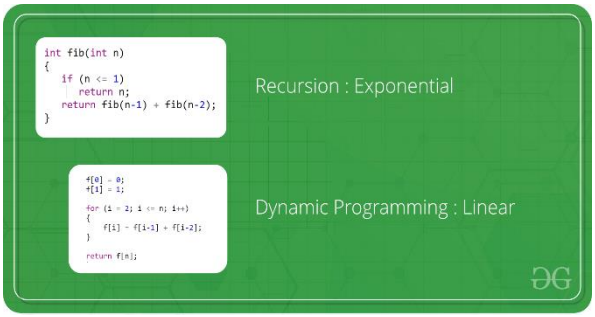
- **Concept of Backtracking**
  - ✓ Understood the backtracking technique, where solutions are built incrementally and discarded when they fail to satisfy constraints**.**

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9	5		8	6 9	3	6		5
1	3					2	5	
							7	4
		5	2		6	3		

- **Rat in a Maze**
  - ✓ Studied solving the Rat in a Maze problem using backtracking, exploring all possible paths to reach the destination.
- **N-Queen Problem**
  - ✓ Mastered solving the N-Queen problem using backtracking, where N queens need to be placed on an NxN chessboard without attacking each other.
- **Sudoku Problem**
  - ✓ Explored solving the Sudoku puzzle using backtracking, understanding how recursive decision-making leads to the correct solution**.**

➢ DYNAMIC PROGRAMMING

- **Introduction**
  - ✓ Understood dynamic programming as a technique to solve complex problems by breaking them down into smaller overlapping subproblems**.**

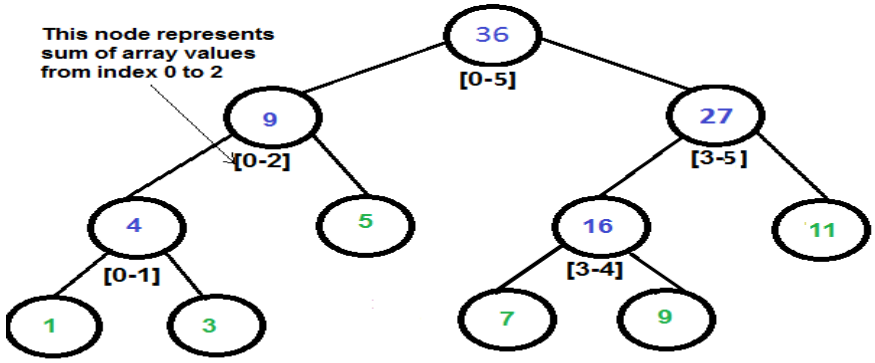


- **Properties of a Dynamic Programming Problem**
  - ✓ Explored the properties of dynamic programming problems, including optimal substructure and overlapping subproblems**.**

- **Overlapping Subproblems Property**
  - ✓ Studied the overlapping subproblems property in dynamic programming, understanding how solutions of smaller instances are reused.
- **Optimal Substructure Property**
  - ✓ Mastered the optimal substructure property, which states that a globally optimal solution can be constructed using optimal solutions to subproblems.
- **Tabulation vs Memoization**
- **Longest Common Subsequence**
  - ✓ Studied finding the longest common subsequence between two sequences, a classic dynamic programming problem with various applications
- **Longest Increasing Subsequence**
- **Maximum Cuts**
- **0-1 knapsack problem**
  - ✓ Studied the 0-1 knapsack problem, where items with weights and values need to be selected optimally to maximize value within a weight limit.

- **TRIE**
  - **Insertion and Search in Trie Data Structure**
  - **Deletion in a Trie**

- **SEGMENT AND BINARY INDEXED TREE**
  - **Introduction**

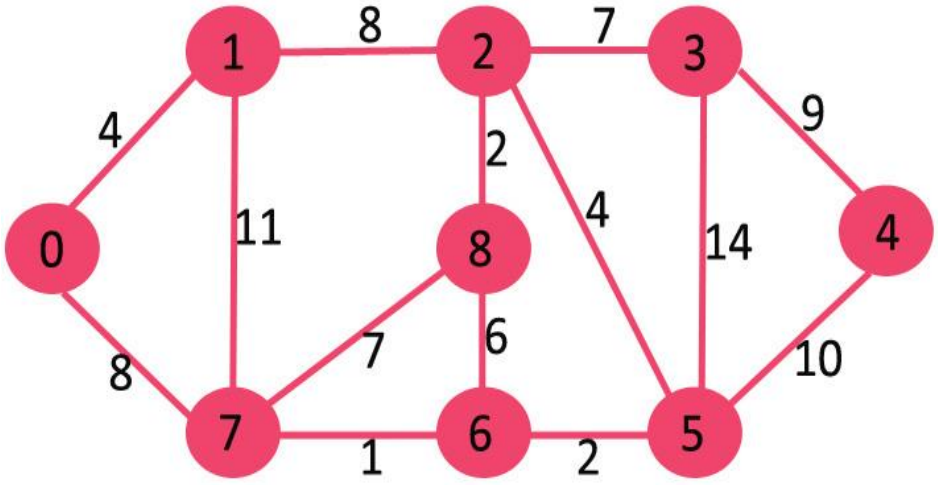


Segment Tree for input array {1, 3, 5, 7, 9, 11}

- **Range Minimum Query**
- **Binary Indexed Tree**

➢ DISJOINT SET

- **Introduction**
  - ✓ Understood the disjoint set data structure, commonly known as the union-find data structure, used for managing disjoint sets of elements.
- **Union by Rank and Path Compression**
  - ✓ Learned about union by rank and path compression techniques used in disjoint set data structures for efficient operations.
- **Kruskal's Minimum Spanning Tree Algorithm**
  - ✓ Studied Kruskal's algorithm, which uses the disjoint set data structure to find the minimum spanning tree of a graph.



Chapter-3 REASON FOR CHOOSING DATA STRUCTURE AND ALGORTIHMS (SELF PACED)

All of the above was part of my summer training during my summer break. I specially choose the Data Structure and Algorithms – Self Paced by GeeksforGeeks. The reason behind choosing mention below:

- ✓ I was interested in Problem Solving and Data Structure and Algorithms since my programming journey.
- ✓ Competitive Programming provides a way to solve the problem in efficient manner and no matter in which language you choose to code.
- ✓ programming challenges are essentially complex problem-solving exercises. Engaging in Data Structure and Algorithm helps individuals sharpen their problem-solving skills, which are valuable not only in computer science but also in many other fields and real-world scenarios.
- ✓ To learn how to make algorithm of a real-life problem that we are facing.
- ✓ It had video lecture of all the topics from which one can easily learn. I prefer learning from video rather than books and notes because sometime we don't understand from books and notes, but video lecture make it easy to understand.
- ✓ It had so many algorithms coding problems with video explained solutions.
- ✓ It had track based learning and weekly assessment to enhance my progress and skill development.
- ✓ It provided weekly live session for building logic and how to approach to given problems.
- ✓ It is life time accessible course to me which I can use to learn after my training also.

Chapter-4 LEARNING OUTCOME

Data Structure and Algorithmis a mind sport, where people find optimal solution and compete against each other to solve some programming questions/logic with an efficient approach and within a time constraint. The goal of Data Structure and Algorithm is to write code to solve a problem within a given timeframe. There are mainly mathematical and logical problems in coding competitions based on Number Theory, Graph Theory, P&C (Permutation and Combination), Games, Geometry, etc. Data Structure and Algorithm provides to solve real-life problems using DSA with an efficient manner. In this we have to solve within the given time complexity and input constraints.

Data Structure and Algorithm is the best way to master your analytical thinking, and logical thinking and improve your coding skills.

DSA is the heart of programming and you cannot ignore it while solving coding problems in competitive programming. Array, Linked List, Stack, Queue, Tree, Trie, Graph, Sorting, Recursion, Dynamic Programming all these basic building blocks of DSA will help you to become a good programmer. The most important thing you need to know what, when and where to apply them. It means which data structure is suitable for what type of problem to get the optimal solution. You should know how to apply a perfect combination of both in the coding problem.

For example: Suppose we have to search employee ID in 5000 pages of Documents how would we do that?

- If we try to search manually or random 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 employee ID is there or not if not, we can move ahead and by repeating this eliminating we can search our employee ID in no time.
  Then we need algorithm to solve this problem in efficiently that is Binary Search Algorithm.

Two reasons to learn Competitive Programming (DSA) –

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

I have learnt a vast number of topics like Trees, Graphs, Arrays, Dynamic Programming, Greedy, Graphs, etc. I understood their basics, their working, their implementation, and their practical use in the problems we face while we solve a problem to write code.

When we work in IT Company, we need to solve the problems, to solving the problems we need some algorithms. When we going to solve the problems, first we divide the problems into small pieces, then develop the algorithm accordingly, Implement in any programming language such as C, C++, Java, etc. The algorithm is based on the data structure and its implementation and working. So, basically, we need to have a good command of DSA to work in IT company.

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, Bin 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.

What I Learned from the course precisely:

- ✓ I Learned Competitive Programming (DSA) 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 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.

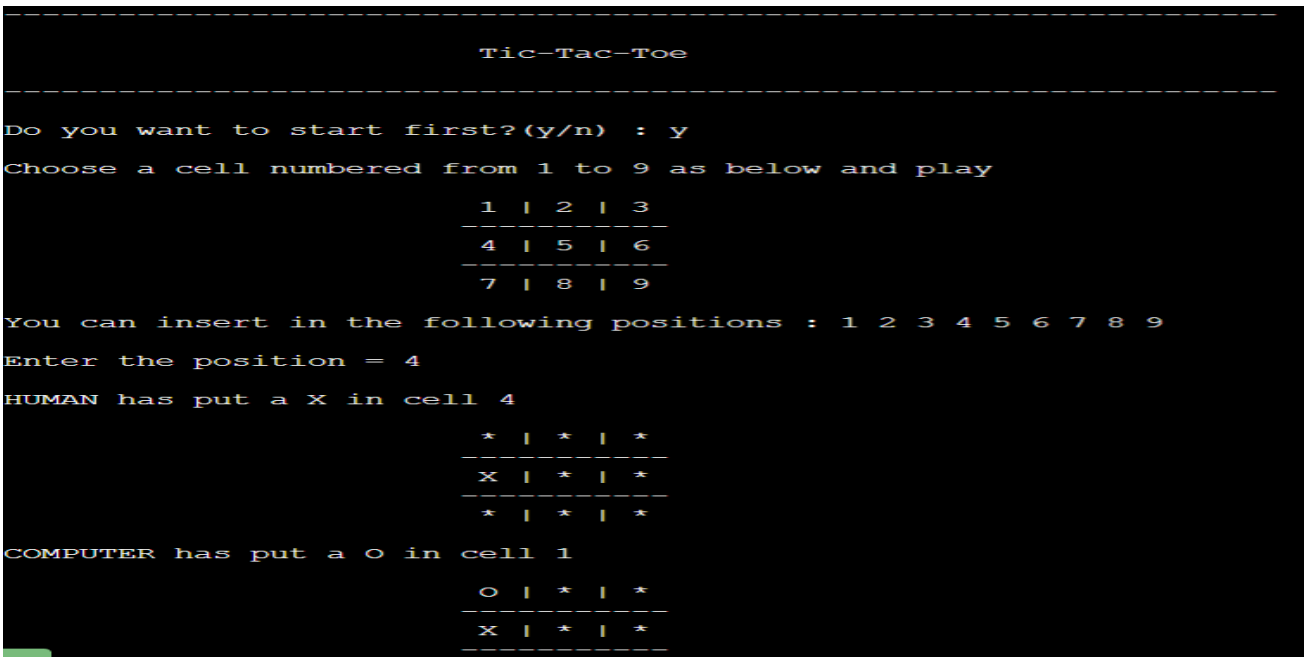
"Data structure and algorithms help in understanding the nature of the problem at a deeper level and thereby a better understanding of the world."

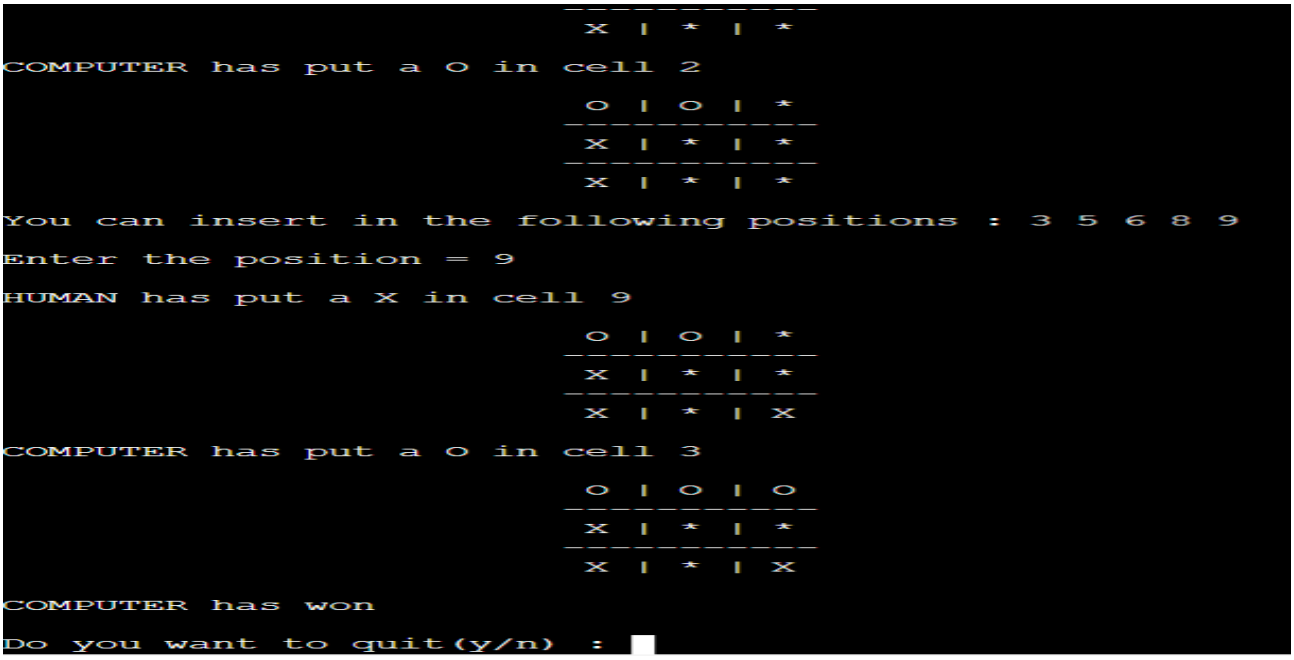
Chapter-5 PROJECT

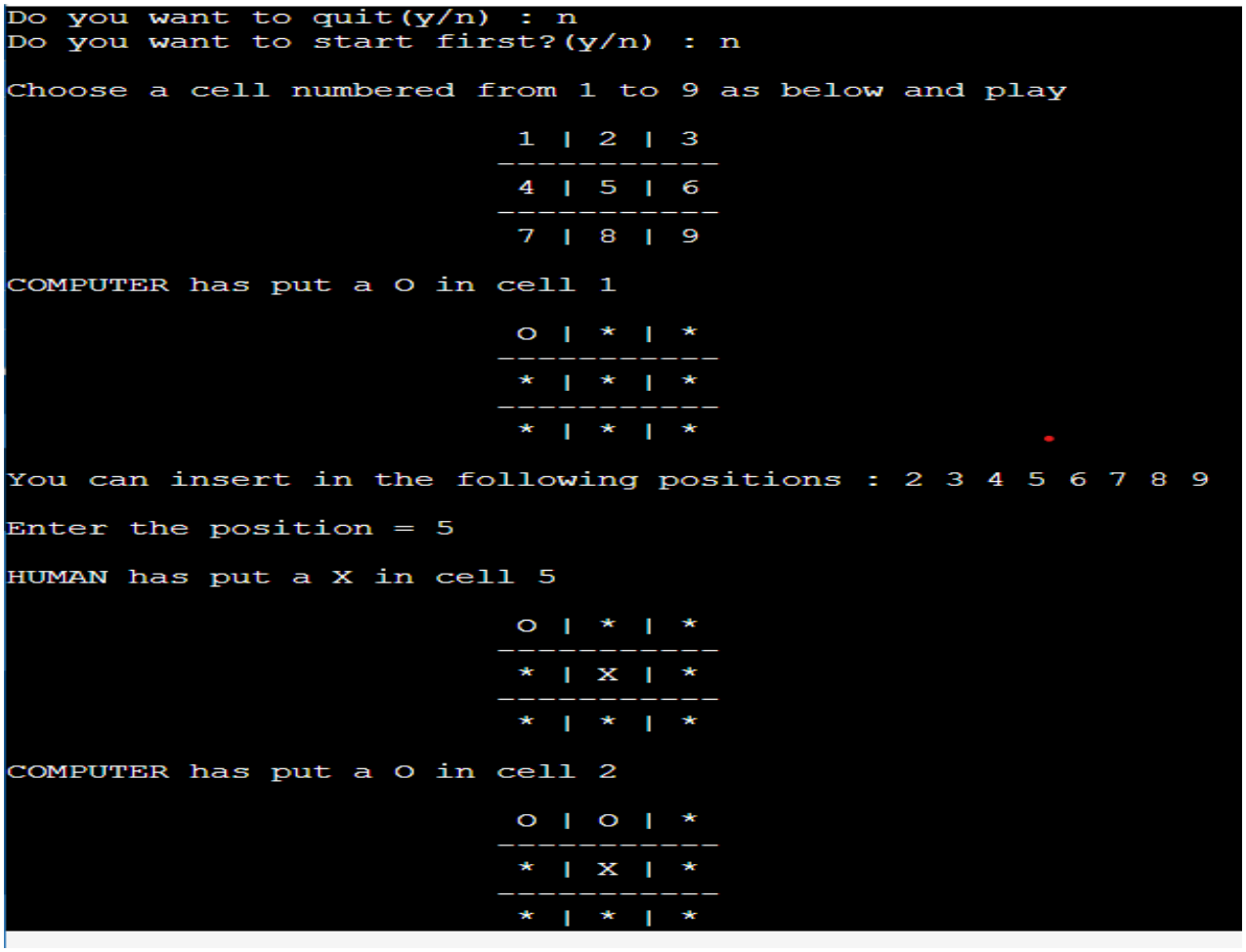
Description of the Game Project:

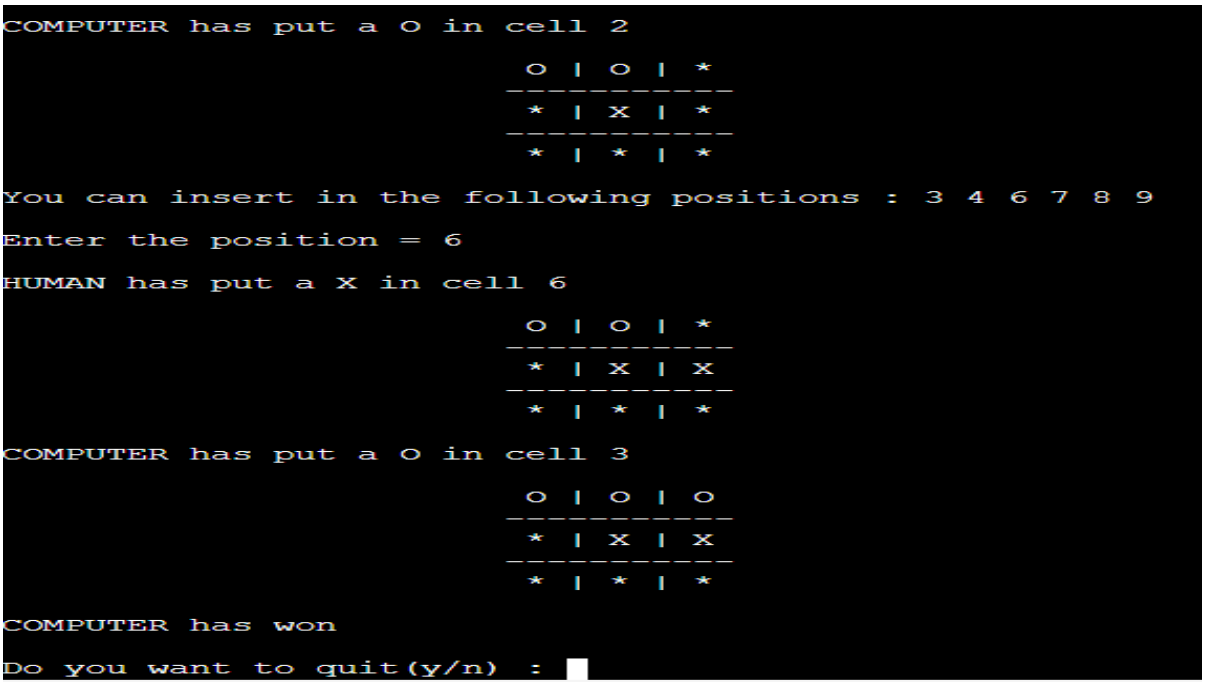
Here I am taking the example of Tic Tac Toe game. I have implemented minimax algorithms with the help of C++ programming language to make it. In this game there are two players one is HUMAN and another one is COMPUTER. There are two filling options in 3X3 grid board with sign of X and O. In this game if HUMAN wants to start or either they can give the opportunity to COMPUTER to start the game. When game will start the first one will move own step and the second opponent will also decide the best move. If component's move fills all the empty boxes like Vertically, Horizontally or Diagonally. Then that the player will be the winner and the opponent will be the looser, if it doesn't satisfy the condition then the game will be draw. As you can see the output below:

Output:



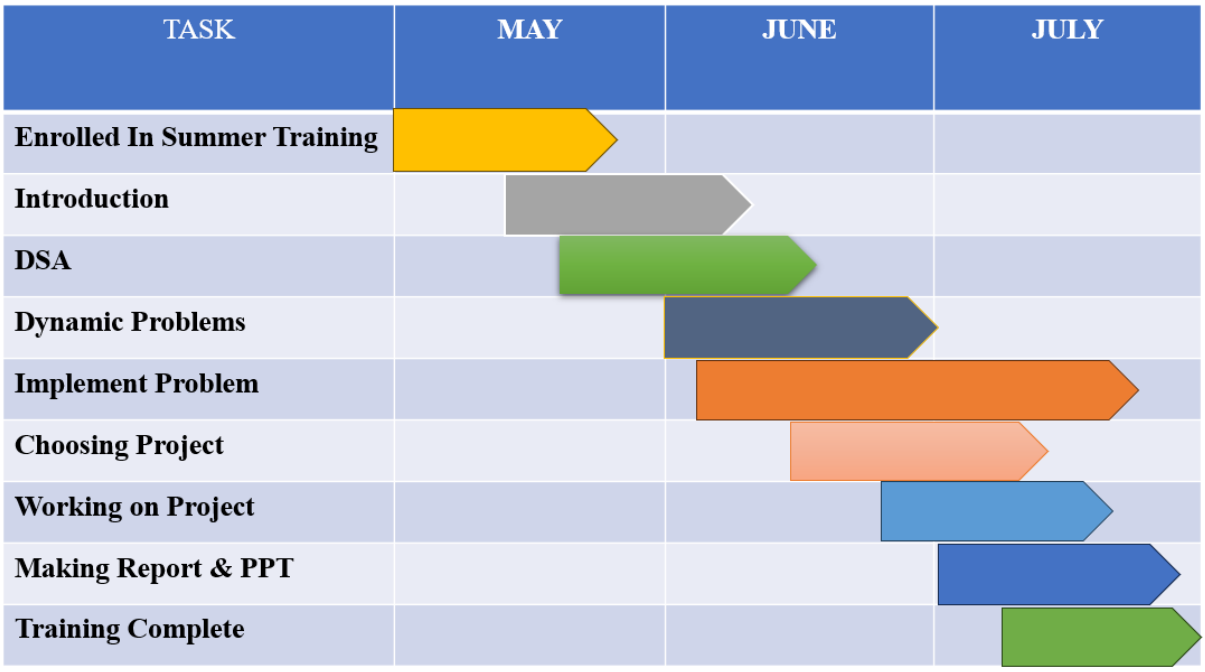

X | * | *
COMPUTER has put a O in cell 2
O | O | *
X | * | *
X | * | *
You can insert in the following positions : 3 5 6 8 9
Enter the position = 9
HUMAN has put a X in cell 9
O | O | *
X | * | *
X | * | X
COMPUTER has put a O in cell 3
O | O | O
X | * | *
X | * | X
COMPUTER has won
Do you want to quit(y/n) :


Do you want to quit(y/n) : n
Do you want to start first?(y/n) : n
Choose a cell numbered from 1 to 9 as below and play
1 | 2 | 3
4 | 5 | 6
7 | 8 | 9
COMPUTER has put a O in cell 1
O | * | *
* | * | *
* | * | *
You can insert in the following positions : 2 3 4 5 6 7 8 9
Enter the position = 5
HUMAN has put a X in cell 5
O | * | *
* | X | *
* | * | *
COMPUTER has put a O in cell 2
O | O | *
* | X | *
* | * | *



GitHub Link for Project: - https://github.com/rohit56900/Tic-Tac-Toe-Game

Chapter-6 GANTT CHART



Final Chapter CONCLUSION AND FUTURE PRESPECTIVE

CONCLUSION:

In conclusion, the journey through the "Data Structure and Algorithms - Self Paced" project on GeeksforGeeks has been nothing short of enlightening. This comprehensive course has provided us with a solid foundation in the world of data structures and algorithms, two fundamental pillars of computer science and software engineering.

Throughout this project, we have delved deep into the intricacies of various data structures, such as arrays, linked lists, trees, graphs, and hash tables. We have also explored a wide range of algorithms, from sorting and searching to dynamic programming and graph traversal. This knowledge is not only crucial for acing technical interviews but also for building efficient and scalable software applications.

One of the standout features of this self-paced project is its accessibility and flexibility. GeeksforGeeks has done an excellent job of structuring the content in a way that caters to learners of all levels. Whether you are a beginner looking to grasp the basics or an experienced developer seeking to refine your skills, this project has something valuable to offer.

The hands-on coding exercises and quizzes provided in this project have been invaluable in reinforcing our understanding. By actively implementing data structures and algorithms in real code, we have gained practical experience that will serve us well in our future endeavors.

Moreover, the GeeksforGeeks platform's vibrant community and discussion forums have allowed us to connect with fellow learners, share insights, and seek help when needed. Learning in such a collaborative environment has enriched our experience and broadened our perspectives.

As we wrap up this project, it's essential to acknowledge that the journey of learning data structures and algorithms is ongoing. The world of technology evolves rapidly, and staying up-to-date with the latest advancements is crucial. Therefore, it is our responsibility to

continue exploring, practicing, and applying what we have learned here in our professional and academic pursuits.

In conclusion, "Data Structure and Algorithms - Self Paced" on GeeksforGeeks has been an enlightening and empowering experience. It has equipped us with essential knowledge and skills that will undoubtedly shape our careers in the field of computer science. Let us carry forward this newfound wisdom and continue our pursuit of excellence in software development and problem-solving. The path ahead is exciting, and armed with the knowledge gained from this project, we are well-prepared to tackle any challenge that comes our way

FUTURE PRESPECTIVE:

The implementation of the Tic-Tac-Toe game within the context of competitive programming opens the door to several exciting future perspectives. As the project evolves and grows, here are some potential avenues for enhancement and expansion:

1. **Smarter Computer Opponent:**
   You can make the computer opponent smarter by using more advanced strategies. Right now, it might be a bit random, but you could teach it to make better moves using techniques like Minimax with Alpha-Beta Pruning. This will make the game more challenging for players.
2. **Customizable Board Size:**
   Extend the game to support customizable board sizes beyond the traditional 3x3 grid. Allowing users to play on larger grids, such as 4x4 or 5x5, introduces additional complexity and strategic possibilities, making the game even more intriguing for competitive programmers.
3. **Enhanced UI/UX:**
   Improve the user interface and experience. Incorporate visually appealing graphics, animations, and user-friendly controls that enhance the overall engagement. A polished interface can make the game more inviting to both participants and spectators, contributing to the competitive atmosphere.
4. **Online Multiplayer Mode:**
   Implement an online multiplayer mode that enables competitive programmers to challenge each other in real-time matches. This could include features like

matchmaking, leaderboards, and chat functionality. Providing a platform for players to compete against opponents from around the world can foster a vibrant competitive community.

5. **Learn How to Play Better:**

   Create a mode where players can practice and learn how to play better. It could start easy and get harder as you go. This way, players can get better at the game by practicing different strategies.

6. **Tournaments and Challenges:**

   Organize regular online tournaments or coding challenges based on the Tic-Tac-Toe game. Set up coding competitions where participants are required to develop AI algorithms that can outwit opponents. This fosters a competitive environment and encourages programmers to innovate their strategies.

7. **Try AI Techniques:**

   Why not let players try out artificial intelligence (AI) techniques? They could use AI libraries to make the computer opponent smarter and learn how AI works.

8. **Cross-Platform Compatibility**

   Ensure that the game is accessible across various platforms, including web, mobile, and desktop. This way, players can play no matter where they are.

Incorporating these future perspectives can transform the Tic-Tac-Toe game within the competitive programming project into a dynamic and engaging platform that challenges programmers to excel in algorithmic thinking and strategic decision-making. By making these changes, the Tic-Tac-Toe game could become even more interesting and fun for competitive programmers. It can bring in more players and create a community of people who love to code and play games.

References

- **GeeksforGeeks Website**
- **Data Structure and Algorithms- Self Paced (GeeksforGeeks)**
- **Google**