**1.INTRODUCTION**

**1.1 Introduction of the System**

**1.1.1Project Title**

Performance analysis and simulation of data structure.

**1.1.2 Category**

Online Simulator

**1.1.3 Overview**

This is an academic application that is used to train the students with data structure. As a result, that can provide a student with better learning opportunities. Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer.In order to solve these problems, this system has been created, that attempts to operate the whole procedure considering the step-by-step approach of algorithm. This method of teaching is an opportunity to make learning engaging, inspiring and fun for students.

**1.2 Background**

**1.2.1 Introduction of the Company**

Not Applicable

**1.2.2 Brief Note on Existing System**

* The existing system consists of manual method that solves the problem using descriptive illustration.
* The performed computation cannot be repeated once it has been erased in the chalk and board system.
* The use of projection devices in the classroom leads to less engaged and interest among the students.
* The delay in the computation leads to time consumption in the classroom.

**1.3 Objectives of the System**

* The main objective of the project is to provide the student with a virtual way of learning the data structure to increase their efficiency in understanding the concept.
* This application enables students to increase their ability in every domain.
* Since the working of different data structure are easily illustrated, this application provides students with quality learning experience with technology’s aid.
* It enables students to analyse and understand the concept in an effective way.

**1.4 Scope of the System**

Scope is the limitation that a process faces from the beginning to the end.

* It best runs in latest version of Windows, Mac, Linux.
* It best runs in latest version of chrome and also runs in other browsers.

**1.5 Structure of the System**

The above system consisting of following modules. The module is the logical separable part of the program or system.

**1.5.1Analysis**

This application generates array and performs different types of searching, sorting, array operation, stack, queue and other operations. It also calculates the time complexity and space complexity of different modules and visualizes the result.

**1.5.2 Module Description**

The performance analysis and simulation of data structure algorithm.

**1.5.2.1 Sorting Module**

In sorting module, we are going to arrange the data in the preferred order. By sorting data, it is easier to search through it quickly and easily.

**1.5.2.1.1 Bubble Sort**

This module compares two adjacent elements and swaps them until they are in order.

**1.5.2.1.2 Selection Sort**

This module finds smallest element and puts in first position. Then finds second smallest element in the list and so on.

**1.5.2.1.3 Insertion Sort**

This module builds the final sorted array on item at time by comparison.

**1.5.2.1.4 Exchange Sort**

This module compares the first element with each following element of the array, making any necessary swaps.

**1.5.2.1.5 Counting Sort**

This module is a collection of objects according to keys that are small positive integers.

**1.5.2.1.6 Heap Sort**

This module creates Min heap or Max heap. It represents the ordering of array in which the root element represents the minimum or maximum element.

**1.5.2.1.7 Merge Sort**

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarrays and then merging the sorted subarray back together to form the final sorted array.

**1.5.2.1.8 Quick Sort**

Quick sort is a divide and conquer algorithm. It picks an element as a pivot and partitions the given array around picked pivot.

**1.5.2.1.9 Brick Sort**

The sorting technique is divided into two phases that is odd-even phases. The algorithm runs until the array elements are sorted and, in each iteration, the two phases occur.

**1.5.2.1.10 Shell Sort**

This sort is highly efficient and is based on the insertion sort algorithm. This algorithm avoids the large shifts as in the insertion sort, where the smaller value is on the far right and must be moved to the far left**.**

**1.5.2.1.11 Bucket sort**

It distributes the element of the array to the number of buckets and then each bucket is individually sorted using different sorting technique or recursively using bucket sort algorithm.

**1.5.2.2 Array Operation Module**

In array operation module, we can hold a fixed number of items and of the same type.

**1.5.2.2.1 Insertion**

This module insertion an element at the given index.

**1.5.2.2.2 Deletion**

This module deletes the specific element.

**1.5.2.3 Searching Module**

In searching module, we are going to find a desired elements from the set of data elements.

**1.5.2.3.1 Linear Search**

This module sequentially checks each element of list until the whole list has been searched.

**1.5.2.3.2 Binary Search**

This module repeatedly divides in half the portion of the list that contain the item until narrowed down to one item.

**1.5.2.3.3 Exponential Search**

This module jumps 2^i elements every iteration where ‘i’ represents the value of loop control variable, and then checks if the search element is present between last jump and the current jump.

**1.5.2.3.4 Interpolation Search**

This module finds a particular item by computing the probe position.

**1.5.2.3.5 Jump Search**

This module checks position of the target data element on a sorted data collection.

**1.5.2.3.6 Ternary Search**

This module determines either that the minimum or maximum cannot be in the first third of the domain or that it cannot be in the last third of the domain, then repeats on the remaining two thirds.

**1.5.2.4 Stack Module**

Stack module, it is a linear data structure that follows LIFO that is Last in First out principle and allows insertion and deletion operation at one end, that is top.

**1.5.2.4.1 Array Implementation**

**1.5.2.4.1.1 Push**

This module puts a new data element on top of the stack.

**1.5.2.4.1.2 Pop**

This module is used to remove data from the stack's top.

**1.5.2.4.2 Linked List Implementation**

**1.5.2.4.2.1 Push**

This module inserts data element on to the list.

**1.5.2.4.2.2 Pop**

This module is used to remove data elements from the list.

**1.5.2.5 Queue Module**

Queue module, it is a collection of items in which the earliest added items can be accessed.

**1.5.2.5.1 Array Implementation**

**1.5.2.5.1.1 Insertion**

In this module the insertion happens at the rear end of the queue

**1.5.2.5.1.2 Deletion**

In this module the deletion happens at the front end of the queue.

**1.5.2.5.2 Linked List Implementation**

**1.5.2.5.2.1 Insertion**

This module adds a new node after the rear and moves the rear to the next node.

**1.5.2.5.2.2 Deletion**

This module removes the front node and moves the front to the next node.

**1.5.2.5.3 Circular queue**

**1.5.2.5.3.1 Insertion**

In this module the insertion happens at the rear end of the queue.

**1.5.2.5.3.2 Deletion**

In this module the deletion happens at the front end of the queue..

**1.5.2.6 Linked list module**

Linked list, is a sequence of data structure which are connected together via links.

**1.5.2.6.1 Singly linked list**

**1.5.2.6.1.1 Insertion at begin**

Node is being added to the beginning of the singly linked list.

**1.5.2.6.1.2 Insertion at end**

Node is being added to the end of the singly linked list.

**1.5.2.6.1.3 Insertion at position**

Node is added at the specified location in the singly linked list.

**1.5.2.6.1.4 Deletion at begin**

Node is deleted from the beginning of the singly linked list.

**1.5.2.6.1.5 Deletion at end**

Node is deleted from the end of the singly linked list.

**1.5.2.6.1.6 Deletion at position**

Delete the element at the specified location in the singly linked list.

**1.5.2.6.1.7 Deletion on element**

Delete the specified element in the singly linked list.

**1.5.2.6.2 Doubly linked list**

**1.5.2.6.2.1 Insertion at begin**

Node is being added to the beginning of the doubly linked list.

**1.5.2.6.2.2 Insertion at end**

Node is being added to the end of the doubly linked list.

**1.5.2.6.2.3 Insertion at position**

Node is added at the specified location in the doubly linked list.

**1.5.2.6.2.4 Deletion at begin**

Node is deleted from the beginning of the doubly linked list.

**1.5.2.6.2.5 Deletion at end**

Node is deleted from the end of the doubly linked list.

**1.5.2.6.2.6 Deletion at position**

Delete the element at the specified location in the doubly linked list.

**1.5.2.6.2.7 Deletion on element**

Delete the specified element in the doubly linked list.

**1.5.2.7 Tree Module**

Tree module, is a hierarchical structure that is used to represent and organise data in a way that is easy to navigate and search.

**1.5.2.7.1 Binary search tree**

**1.5.2.7.1.1 Insertion**

Insert a node in the tree.

**1.5.2.7.1.2 Deletion**

Deletes a node from the tree.

**1.5.2.7.1.3 Searching**

Searches for a node in the tree.

**1.5.2.7.1.4 Pre-order**

Pre-order traversal of the tree.

**1.5.2.7.1.5 Post-order**

Post-order traversal of the tree.

**1.5.2.7.1.6 In-order**

In-order traversal of the tree.

**1.5.2.8 Graph Module**

Graph module, is a non-linear data structure consisting of vertices and edges.

**1.5.2.8.1 DFS**

Depth-First-Search is an algorithm for traversing or searching graph data structure. The algorithm starts at root node and explores as far as possible along each branch before backtracking.

**1.5.2.8.2 BFS**

Breadth-First-Search is the graph traversal approach, starts at a source node and layer through the graph, analysing the nodes directly related to the source node.

**1.6 System architecture**

**USERS**

User level

**GUI**

Tools

Buttons

Console

Commands

Menus

Application level

**DATASTRUCTURE APPLICATION**

Stack

Searching

Array Operation

Sorting

Queue

Graph

Tree

Linked List

Web Browser

Systemlevel

Data Structure tools JavaScript HTML CSS

Operating System

Processing Hardware

Fig 1.1 SYSTEM ARCHITECTURE

* 1. **End user**
* Staff
* Student

**1.8 Software/Hardware used for development**

**1.8.1 Software**

* Operating system: Windows 10 or above
* CSS
* Java script
* Html
* Web Browser (Chrome, Yahoo etc)

**1.8.2 Hardware**

* Processor: Intel Core i3 and above
* RAM: 4GB
* Hard Disk: 500GB

**1.9 Software/Hardware used for implementation**

**1.9.1 Software**

* Operating system: Windows 10 or above
* CSS
* Java script
* Html
* Web Browser (Chrome, Yahoo etc)

**1.9.2 Hardware**

* Processor: Intel Core i3 and above
* RAM: 4GB
* Hard Disk: 500GB