**1.INTRODUCTION**

**1.1 Introduction of the System**

**1.1.1Project Title**

Performance analysis and simulation of data structure algorithms.

**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 algorithms.

**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 subarray 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**

**GUI**

Menus

Commands

Console

Tools

Buttons

**DATASTRUCTURE APPLICATION**

Sorting

Array Operation

Searching

Stack

Queue

Linked List

Tree

Graph

Web Browser

Data Structure tools JavaScript HTML CSS

Operating System

Processing Hardware

User level

Application level

System level

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

**2.SOFTWARE REQUIREMENT SPECIFICATION**

**2.1 Introduction**

IEEE defines Software Requirement Specification as a condition or a capability must be met or possess by a system to satisfy a contract, standard, specification or other formally imposed document. in other words, the SRS is the step that one would use to find the path to getting one’s own goal.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. It defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

SRS can help developers reduce the time and effort necessary to meet their goals as well as save money on the cost of development.

**2.2 Overall Description**

**2.2.1 Product perspective**

Product perspective is essentially the relationship of the product to other products. It is not the part of any other product.

**2.2.1.1 System Interface**

This application runs in the latest version of Chrome or Firefox browser on Windows, Linux and Mac.

**2.2.1.2 User Interface**

This application GUI provides buttons, labels, textbox, scrollbars allowing for easy control by a keyboard and mouse.

**2.2.1.3 Hardware Interface**

Not applicable.

* + - 1. **Software Interface**
* This application uses HTML and Java Script.
* This product uses browser and operates on Windows XP or higher version.

**2.2.1.5 Communication Interface**

This application allows to interface LAN, WAN, MAN, Internet.

**2.2.1.6 Interface with Server**

Not applicable.

**2.2.2 Product Functions**

The project has analysis on data structure and it provides summary of the function to be performed by the software.

* The module displays the given data in an order and interactively shows the result after specific operation like sorting, searching, stack, queue, linked list, tree and graph.
* It includes outcome of each operation in the form of time complexity and space complexity exhibits the result.
* It will visualize and simulate Various Data Structure Algorithms.

**2.2.3 User Characteristics**

Specify the End Users minimum characteristics or knowledge required use the application.

* To use this application user should require basic knowledge of using the computer operation.
* The displayed result should be understood by the applicant.
* End User should have the knowledge of Data Structure Concepts.

**2.2.4 General constraints**

General Constraint describe how the product operates inside various circumstances and limit the options designers have if building the product.

* The software runs on the windows 10 or any high version of operating system
* It best runs on the latest version of chrome and also runs in other web browsers like edge, Firefox etc.
* This application can be run within the minimum amount of battery and storage.

**2.2.5 Assumptions and Dependencies**

These factors are not designing constraint on the software but any changes to these factors can affect the requirement in the SRS.

* For windows 10, system requires 1GHz or faster with 2 or more cores.
* System should have minimum of 4GB RAM with 64 GB ROM.

**2.3 Special requirements**

Not applicable

**2.4 Functional Requirements**

This section gives the functional capabilities of the system that specifying the input, desired output and processing requirement.

**2.4.1 Sorting module**

**2.4.1.1 Bubble Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and compares two adjacent elements and swaps them until they are in order.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.2 Selection sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers then finds smallest element and puts in first position. Then finds second smallest element in the list and so on until they are in order.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.3 Insertion Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and compares the selected element with the previous array elements and then displays the elements in the sorted order.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.4 Exchange Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and compares the first element with each following element of the array and then displays the elements in the sorted order.
3. **Output**: Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.5 Counting Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and sorts the array also counts the number of reptation of array elements.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.6 Heap Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers later it divides the sorted and unsorted region and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into sorted region.
3. **Output**: Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.7 Bucket Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and distributes the elements of array in number of buckets and each of the bucket is individually sorted using different sorting technique or recursively using bucket sort algorithm.
3. **Output**: Displays the sorted bar graph also calculates the time and space complexity.
   * + 1. **Merge Sort**
4. **Input**: Array of numbers
5. **Process**: This sorting technique takes the array of numbers and then starts dividing an array into smaller subarrays, sorting each subarray and then merging the sorted subarray back together to form the final sorted array.
6. **Output**: Displays the sorted bar graph.

**2.4.1.9 Quick Sort**

1. **Input:** Array of numbers
2. **Process:** This sorting technique takes the array of numbers and then picks an element as a pivot and partitions the given array around picked pivot.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.
   * + 1. **Brick Sort**
4. **Input:** Array of numbers
5. **Process**: This sorting technique takes the array of numbers it functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order the elements are switched. It repeats this step for even/odd indexed pair. Then it alternates between the odd/even and even/odd steps until the list is sorted.
6. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.1.11 Shell Sort**

1. **Input**: Array of numbers
2. **Process:** This sorting technique takes the array of elements it starts by pairs of elements far apart from each other, then progressively reducing the gap between the elements to be compared.
3. **Output:** Displays the sorted bar graph also calculates the time and space complexity.

**2.4.2 Array Operation module**

**2.4.2.1 Insertion**

1. **Input:** Array of numbers, number to be inserted, position of inserted number.
2. **Process:** This operation takes the array of numbers and checks the position in the linear order then inserts the specified element at specified position.
3. **Output:** Displays the array with inserted element using bar graph also calculates the time and space complexity.

**2.4.2.2 Deletion**

1. **Input:** Array of numbers, number to be deleted.
2. **Process:** This operations takes the array of numbers and checks in the linear order then deletes the specified element.
3. **Output:** Displays the final bar graph excluding the deleted element and then displays the position of the element deleted also calculates the time and space complexity.

**2.4.3 Searching module**

**2.4.3.1 Linear search**

1. **Input:** Array of numbers, element to be searched.
2. **Process**: This searching technique takes the array of numbers and searches the element linearly if found displays the position of element found.
3. **Output**: Displays the searched element in the bar graph with different colour and also displays the position of the element found, time and space complexity.

**2.4.3.2 Binary search**

1. **Input**: Array of numbers, element to be searched.
2. **Process:** This technique initially sorts the given array if searched element is greater than middle element then it searches at right side otherwise left.
3. **Output**: Displays the searched element in the bar graph with different colour and also displays the position of the element found, time and space complexity.

**2.4.3.3 Exponential search**

1. **Input:** Array of numbers, key element.
2. **Process:** It searches for an element in a sorted array by jumping 2^I elements every iteration where I represents the value of loop control variable, and then verifying if search element is present between last jump and current jump.
3. **Output:** Displays the searched element along the sorted bar graph also the time and space complexity.

**2.4.3.4 Interpolation search**

1. **Input:** Array of numbers, key element
2. **Process:** Interpolation search is an algorithm for searching for a key in an array that has been ordered by numerical values assigned to the keys (key values).
3. **Output:** Displays the searched element along the sorted bar graph also the time and space complexity.

**2.4.3.5 Jump search**

1. **Input:** Array of numbers, key element
2. **Process:** Jump search technique also works for ordered lists. It creates a block and tries to find the element in that block. If the item is not in the block, it shifts the entire block. The block size is based on the size of the list. If the size of the list is n then block size will be √n.
3. **Output**: Displays the searched element along the sorted bar graph also the time and space complexity.

**2.4.3.6 Ternary search**

1. **Input**: Array of numbers, key element
2. **Process**: Here the array elements are divided into 3 groups within the key element is searched.
3. **Output**: Displays the searched element along the sorted bar graph also the time and space complexity.

**2.4.4 Stack**

**2.4.4.1 Array Implementation**

**2.4.4.1.1 Push**

1. **Input:** Number
2. **Process:** It stores the number at the beginning of the container and checks whether the stack is full.
3. **Output**: Enters the number within the container also calculates time and space complexity.

**2.4.4.1.2 Pop**

1. **Input:** Click on POP Button.
2. **Process**: It removes the last entered number and checks whether the stack empty.
3. **Output**: Removes the number from the container also calculates time and space complexity.

**2.4.4.2 Linked List Implementation**

**2.4.4.2.1 Push**

1. **Input:** Number
2. **Process**: It creates the node first and allocates memory if the l list is empty then the node is pushed as the first element of the list otherwise link the previous element to the address field of new node.
3. **Output:** Pushes the element to the list also calculates time and space complexity.

**2.4.4.2.2 Pop**

1. **Input:** click on pop button.
2. **Process:** Node is removed from the end of the linked list. Therefore, must delete the value stored in the head pointer. Then the following node will become the head node.
3. **Output:** Removes the element to the list also calculates time and space complexity.

**2.4.5 Queue**

**2.4.5.1 Array Implementation**

**2.4.5.1.1 Insertion**

1. **Input:** Number
2. **Process:** It stores the number at the beginning of the container and checks whether the queue is full.
3. **Output**: Enters the number within the container also calculates time and space complexity.

**2.4.5.1.2 Deletion**

1. **Input:** click the delete button.
2. **Process**: It removes the first entered number and checks whether the queue empty.
3. **Output**: Removes the number from the container also calculates time and space complexity.

**2.4.5.2 Linked List Implementation**

**2.4.5.2.1 Insertion**

1. **Input:** Number
2. **Process:** It adds an element to the end of the queue. The new element becomes the last element of the queue.
3. **Output**: Enters the element to the list also calculates time and space complexity.

**2.4.5.2.2 Deletion**

1. **Input:** click the delete button.
2. **Process:** Removes the element which was first inserted in the queue.
3. **Output**: Removes the element from the list also calculates time and space complexity.
   * + 1. **Circular Queue**

**2.4.5.3.1 Insertion**

1. **Input:** Number
2. **Process:** It stores the number at the beginning of the circle and checks whether the circular queue is full.
3. **Output**: Enters the number in the circle also calculates time and space complexity.

**2.4.5.3.2 Deletion**

1. **Input:** click the delete button.
2. **Process:** It removes the first entered number and checks whether the circular queue empty.
3. **Output:** Removes the number in the circle also calculates time and space complexity.
   * 1. **Linked list**
        1. **Singly linked list** 
           1. **Insertion at begin**
4. **Input**: Node elements
5. **Process:** Insert the node at the beginning of the Singly linked list.
6. **Output**: Displays the Singly linked list with time and space complexity.
   * + - 1. **Insertion at end**
7. **Input:** Node elements
8. **Process:** Insert the node at the end of the Singly linked list.
9. **Output:** Displays the Singly linked list with time and space complexity.
   * + - 1. **Insertion at position**
10. **Input:** Node elements, position.
11. **Process**: Insert the node at the specific position of the Singly linked list.
12. **Output**: Displays the Singly linked list with time and space complexity.
    * + - 1. **Deletion at begin**
13. **Input:** Node elements
14. **Process:** Delete the node at the beginning of the Singly linked list.
15. **Output:** Displays the Singly linked list with time and space complexity.
    * + - 1. **Deletion at end**
16. **Input:** Node elements
17. **Process:** Delete the node at the end of the Singly linked list.
18. **Output:** Displays the Singly linked list with time and space complexity.
    * + - 1. **Deletion at position**
19. **Input:** Node position.
20. **Process:** Delete the node at specific location of the Singly linked list.
21. **Output:** Displays the Singly linked list with time and space complexity.
    * + - 1. **Deletion on element**
22. **Input:** Element
23. **Process:** Delete the specific element of the Singly linked list.
24. **Output:** Displays the Singly linked list with time and space complexity.

**2.4.6.2 Doubly linked list**

* + - * 1. **Insertion at begin**

1. **Input:** Node elements
2. **Process:** Insert the node at the beginning of the Doubly linked list.
3. **Output:** Displays the Doubly linked list with time and space complexity.
   * + - 1. **Insertion at end**
4. **Input:** Node elements
5. **Process:** Insert the node at the end of the Doubly linked list.
6. **Output:** Displays the Doubly linked list with time and space complexity.
   * + - 1. **Insertion at position**
7. **Input:** Node elements, position.
8. **Process:** Insert the node at the specific position of the Doubly linked list.
9. **Output:** Displays the Doubly linked list with time and space complexity.
   * + - 1. **Deletion at begin**
10. **Input:** Node elements.
11. **Process:** Delete the node at the beginning of the Doubly linked list.
12. **Output:** Displays the Doubly linked list with time and space complexity.
    * + - 1. **Deletion at end**
13. **Input:** Node elements
14. **Process:** Delete the node at the end of the Doubly linked list.
15. **Output:** Displays the Doubly linked list with time and space complexity.
    * + - 1. **Deletion at position**
16. **Input:** Node position.
17. **Process:** Delete the node at specific location of the Doubly linked list.
18. **Output**: Displays the Doubly linked list with time and space complexity.
    * + - 1. **Deletion on element**
19. **Input:** Element
20. **Process:** Delete the specific element of the Doubly linked list.
21. **Output:** Displays the Doubly linked list with time and space complexity.

**2.4.7 Tree**

**2.4.7.1 Binary Search Tree**

**2.4.7.1.1 Insertion**

1. **Input:** Number
2. **Process:** This technique is used to add new element to the binary search tree in appropriate location.
3. **Output:** Displays the binary tree including the new element also calculates the time and space complexity.

**2.4.7.1.2 Deletion**

1. **Input:** Number
2. **Process:** This technique will delete the specified element from a binary search tree.
3. **Output:** Displays the binary tree excluding the removed element also calculates the time and space complexity.

**2.4.7.1.3 Searching**

1. **Input:** key element
2. **Process:** This technique is used for locating a key from within a set.
3. **Output:** Displays the message found if element is found otherwise displays not found message and also calculates the time and space complexity.
   * + - 1. **Preorder**
4. **Input:** List of alphabets.
5. **Process:** This technique that traverses from root to the left subtree then to the right subtree.
6. **Output:** Displays the list of traversed order.

**2.4.7.1.5 post-order**

1. **Input:** List of alphabets
2. **Process:** This technique that traverses from left subtree to the right subtree then to the root.
3. **Output:** Displays the list of traversed order.

**2.4.7.1.6 In-order**

1. **Input**: List of alphabets.
2. **Process:** This technique that traverses from left subtree to the root then to the right subtree.
3. **Output:** Displays the list of traversed order.

**2.4.8 Graph**

**2.4.8.1 BFS**

1. **Input:** Enter the number of vertices, matrix element and source vertex.
2. **Process**: BFS or Breadth-First-Search starts from the top node in the graph and travels down until it reached the root node.
3. **Output:** It will display the complete vertex sequence from Source to Destination Node in given tree.
   * + 1. **DFS**
   1. **Input**: Enter the number of vertices, matrix element and source vertex.
   2. **Process:** DFS or Depth-First-Search starts from the top node and follows the path to reaches the end node of the path.
   3. **Output:** It will display the complete vertex sequence from Source to Destination Node in given tree.

**2.5 Design Constraints**

It should include standard compliance that enforces restrictions on designer.

* + 1. **Hardware Constraints**
* Ram - 4GB
* Hard Disk 500GB
  + 1. **Software Constraints**
* Web browser (for example: Chrome, edge etc.)
  + 1. **Fault Tolerance**

Ability of the computer system to continue operating without interruption when one or more of its components fail. Fault Tolerance is achieved by every Input data. If the data is failed during its validation, then that input will be discarded. Only the correct information will be sent.

* + 1. **Security**

Not applicable

* + 1. **Standard Compliance**

It is a graphical user interface which is created by using HTML and CSS. We can navigate through each data structure button and can perform each operation.

* 1. **System Attributes**

The system attribute specifies overall attributes that system should have.

* **Reliability:**

The application system must be highly reliable and it should generate all the output in the correct order.

* **Portability:**

In relation to the software, is a measure of how easily an application can be transferred from one computer environment to another.

* **Maintainability:**

The application should be easy to maintain. The code should be written in a way that it favours implementation of function.

* **Scalability:**

The application system does not undergo any problem when multiple operations perform simultaneously.

* 1. **Other requirements**

Not applicable

**3.SYSTEM DESIGN**

**3.1 Introduction**

* System design is the process of defining the architecture, module interfaces and data for a system to satisfy the specified requirements.
* The purpose of the design phase is to plan the solution of the problem specified by the requirement documents.
* This is the first step that moving from problem domain to the solution domain.
* The design of the system is essentially a blueprint or a plan for a solution for the system.

**3.2 Assumptions and Constraints**

An assumption is a condition you think to be true and constrain is fixed limitations of project development.

* All the functional requirement collected from client are sufficient for the project life-cycle.
* All the Non-functional and Specific requirement specified in SRS well enough for the development of system.
* This software is to be completed with time constraint of 3 months.

**3.3 Functional decomposition**

Functional decomposition is the process of taking a complex process and breaking it down into its smaller, simpler parts. Using Functional decomposition larger or complex functionalities are more easily understood. It is mainly used during project analysis phase, so each phase can be viewed as software. So, this has modular with some sub modules.

**3.3.1 System Software Architecture**

**PERFORMANCE ANALYSIS AND SIMULATION OF DATA STRUCTURE ALGORITHMS**

A

B

C

E

F

G

H

Sorting

Array Operation

Searching

Stack

Queue

Linked List

Tree

Graph

Selection Sort

Insertion Sort

Exchange Sort

Counting Sort

Heap Sort

Radix Sort

Merge Sort

Quick Sort

Brick Sort

Shell Sort

D

A

Bubble Sort

Selection Sort

Insertion Sort

Exchange Sort

Counting Sort

Heap Sort

Bucket Sort

Merge Sort

Quick Sort

Brick Sort

Shell Sort

B

Insertion

Deletion

C

Linear Search

Binary Search

Exponential Search

Interpolation Search

Jump Search

Ternary Search

D

Array implementation

Push

Pop

Linked list Implementation

Push

Pop

E

Array implementation

Insertion

Deletion

Linked list Implementation

Insertion

Deletion

F

Singly Linked List

Doubly Linked List

1

2

1

Insertion at begin

Insertion at end

Insertion at position

Deletion at begin

Deletion at end

Deletion at position

2

Insertion at begin

Insertion at end

Insertion at position

Deletion at begin

Deletion at end

Deletion at position

G

Binary Search Tree

Insertion

Deletion

Searching

Pre-order

Post-order

In-order

H

BFS

DFS

Fig 3.1 System Software Architecture

**3.3.2 System Technical Architecture**

User

Internet

Web browser

Application

Fig 3.2 System Technical Architecture

**3.3.3 System Hardware Architecture**

Processing Hardware

Computer

Mass Storage

Display

Input

Fig 3.3 System Hardware Architecture

**3.3.4 External Interfaces**

Not Applicable.

**3.4 Description of Programs**

**3.4.1 Context Flow Diagram (CFD)**

In CFD entire system is considered as a single process. Context flow diagram shows input and outputs of the system. It shows all the external entities that interact with the system and how the data flows between these external entities and system.

Teacher/student

Requesting for array insertion and deletion

Displays the resultant array

Requesting for visualization Sorting

Teacher/Student

Requesting for visualization Searching

Requesting for stack push and Pop

Displays the resultant stack

Display searched element in array

Display sorted array

Teacher/student

Display resultant linked list

Teacher/Student

Requesting for queue insertion and deletion

Requesting for visualization of linked-list

Display resultant tree

Requesting for visualization of tree

Requesting for visualization of graph

Display resultant graph

Display resultant queue

Fig 3.4 Context Flow Diagram

**3.4.2 Data Flow Diagram (DFDs Level 0, Level 1, Level 2)**

Data flow diagram shows the flow of data through system. Data flow diagrams also called the data flow graphs. It views a system as a function that transforms the inputs into desired outputs. It aims to capture the transformation that taken place within a system to the input data so that eventually the output data is produced.

|  |  |  |
| --- | --- | --- |
| **Symbols** | **Name** | **Description** |
|  | **Process** | **It performs transformation of data from one state to another.** |
|  | **Source /Sink** | **It represents the external entity that may be either source or Sink.** |
|  | **Flow of data** | **It represents the flow of data from source to destination** |
|  | **Data Source/Data storage** | **It is the place where data is stored.** |

Table 3.1 Data Flow Diagram Symbols.

**3.4.3 LEVEL 0**

USER

fig 3.5 DFD for modules (Level 0)

**3.5 Description of the Component**

**3.5.1 Sorting module**

**LEVEL-1**

Teacher/Staff

Fig 3.6 Sorting (Level-1)

**3.5.1.1 Bubble Sort**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.7 Bubble Sort

**3.5.1.1.1 Input**

Array of numbers

**3.5.1.1.2 Process**

This sorting technique takes the array of numbers and compares two adjacent elements and swaps them until they are in order.

**3.5.1.1.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.1.4 Interface with other functional component**

Not applicable.

**3.5.1.1.5 Resource allocation**

style.css file.

**3.5.1.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.2 Selection Sort**

Teacher/ student

FALSE

TRUE

**LEVEL-2**

Fig 3.8 Selection Sort

**3.5.1.2.1 Input**

Array of numbers

**3.5.1.2.2 Process**

This sorting technique takes the array of numbers then find smallest element and puts in first position. Then finds second smallest element in the list and so on until they are in order.

**3.5.1.2.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.2.4 Interface with other functional component**

Not applicable

**3.5.1.2.5 Resource allocation**

style.css file.

**3.5.1.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.3 Insertion Sorting**

Teacher/ student

FALSE

TRUE

**LEVEL-2**

Fig 3.9 Insertion Sorting

**3.5.1.3.1 Input**

Array of numbers

**3.5.1.3.2 Process**

This sorting technique takes the array of numbers and compares the selected element with the previous array elements and then displays the elements in the sorted order.

**3.5.1.3.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.3.4 Interface with other functional component**

Not applicable

**3.5.1.3.5 Resource allocation**

style.css file.

**3.5.1.3.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.4 Exchange Sorting**

Teacher/ student

Fig 3.10 Exchange Sort

**LEVEL-2**

FALSE

TRUE

**3.5.1.4.1 Input**

Array of numbers

**3.5.1.4.2 Process**

This sorting technique takes the array of numbers and compares the first element with each following element of the array and then displays the elements in the sorted order.

**3.5.1.4.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.4.4 Interface with other functional component**

Not applicable

**3.5.1.4.5 Resource allocation**

style.css file.

**3.5.1.4.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.5 Counting Sorting**

FALSE

TRUE

Teacher/ student

Fig 3.11 Counting Sort

**LEVEL-2**

**3.5.1.5.1 Input**

Array of numbers

**3.5.1.5.2 Process**

This sorting technique takes the array of numbers and sorts the array also counts the number of repetitions of array elements.

**3.5.1.5.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.5.4 Interface with other functional component**

Not applicable

**3.5.1.5.5 Resource allocation**

style.css file.

**3.5.1.5.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.6 Heap Sorting**

Teacher/ student

Fig 3.12 Heap Sort

**LEVEL-2**

FALSE

TRUE

**3.5.1.6.1 Input**

Array of numbers

**3.5.1.6.2 Process**

This sorting technique takes the array of numbers later it divides the sorted and unsorted region and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into sorted region.

**3.5.1.6.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.6.4 Interface with other functional component**

Not applicable

**3.5.1.6.5 Resource allocation**

style.css file.

**3.5.1.6.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.7 Bucket Sorting**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.13Bucket Sort

**3.5.1.7.1 Input**

Array of numbers

**3.5.1.7.2 Process**

It works by distributing an element of an array into a number of buckets. Each bucket is then sorted individually either using a different sorting algorithm or by recursively applying the bucket sorting algorithm.

**3.5.1.7.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.7.4 Interface with other functional component**

Not applicable

**3.5.1.7.5 Resource allocation**

style.css file.

**3.5.1.7.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.8 Merge Sorting**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.14 Merge Sort

**3.5.1.8.1 Input**

Array of numbers

**3.5.1.8.2 Process**

This sorting technique takes the array of numbers and compares the selected element with the previous array elements and then displays the elements in the sorted order.

**3.5.1.8.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.8.4 Interface with other functional component**

Not applicable

**3.5.1.8.5 Resource allocation**

style.css file.

**3.5.1.8.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.9 Quick Sorting**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.15 Quick Sort

**3.5.1.9.1 Input**

Array of numbers.

**3.5.1.9.2 Process**

This sorting technique takes the array of numbers and then picks an element as a pivot and partitions the given array around picked pivot.

**3.5.1.9.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.9.4 Interface with other functional component**

Not applicable

**3.5.1.9.5 Resource allocation**

style.css file.

**3.5.1.9.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.10 Brick Sorting**

Teacher/ student

FALSE

TRUE

**LEVEL-2**

Fig 3.16 Brick Sort

**3.5.1.10.1 Input**

Array of numbers

**3.5.1.10.2 Process**

This sorting technique takes the array of numbers it functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order the elements are switched. It repeats this step for even/odd indexed pair. Then it alternates between the odd/even and even/odd steps until the list is sorted.

**3.5.1.10.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.10.4 Interface with other functional component**

Not applicable

**3.5.1.10.5 Resource allocation**

style.css file.

**3.5.1.10.6 User interface**

Buttons, labels, textbox, message box.

**3.5.1.11 Shell Sorting**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.17 Shell Sort

**3.5.1.11.1 Input**

Array of number.

**3.5.1.11.2 Process**

This sorting technique takes the array of elements it starts by pairs of elements far apart from each other, then progressively reducing the gap between the elements to be compared.

**3.5.1.11.3 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**3.5.1.11.4 Interface with other functional component**

Not applicable

**3.5.1.11.5 Resource allocation**

style.css file.

**3.5.1.11.6 User interface**

Buttons, labels, textbox, message box.

**3.5.2 Array Operation module**

**LEVEL-1**

Teacher/Staff

Fig 3.18 Array Operation (Level-1)

**3.5.2.1.1 Insertion**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.19 Array Insertion

**3.5.2.1.1 Input**

Array of numbers, number to be inserted, position of inserted number.

* + - * 1. **Process**

This operation takes the array of numbers and checks the position in the linear order then inserts the specified element at specified position.

**3.5.2.1.3 Output**

Displays the array with inserted element using bar graph also calculates the time and space complexity.

**3.5.2.1.4 Interface with other functional component**

Not applicable

**3.5.2.1.5 Resource allocation**

style.css file.

**3.5.2.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.2.2.1 Deletion**

Teacher/ student

FALSE

TRUE

Fig 3.20 Array Deletion.

**LEVEL-2**

s

**3.5.2.2.1 Input**

Array of numbers, number to be deleted.

**3.5.2.2.2 Process**

This operations takes the array of numbers and checks in the linear order then deletes the specified element.

**3.5.2.2.3 Output**

Displays the final bar graph excluding the deleted element and then displays the position of the element deleted also calculates the time and space complexity.

**3.5.2.2.4 Interface with other functional component**

Not applicable

**3.5.2.2.5 Resource allocation**

style.css file.

**3.5.2.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3 Searching module**

Teacher/Staff

**LEVEL-1**

Fig 3.21 Searching Operation (Level-1)

**3.5.3.1 Linear Search**

**LEVEL-2**

Teacher/ student

FALSE

TRUE

Fig 3.22 Linear Search

**3.5.3.1.1 Input**

Array of numbers, element to be searched.

**3.5.3.1.2 Process**

This searching technique takes the array of numbers and searches the element linearly if found displays the position of element found.

**3.5.3.1.3 Output**

Displays the searched element in the bar graph with different colour and also displays the position of the element found, time and space complexity.

**3.5.3.1.4 Interface with other functional component**

Not applicable

**3.5.3.1.5 Resource allocation**

style.css file.

**3.5.3.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3.2 Binary search**

Teacher/ student

FALSE

TRUE

**LEVEL-2**

Fig 3.23 Binary Search

**3.5.3.2.1 Input**

Array of numbers, key element

**3.5.3.2.1 Process**

This technique initially sorts the given array if searched element is greater than middle element then it searches at right side otherwise left.

**3.5.3.2.3 Output**

Displays the searched element in the bar graph with different colour and also displays the position of the element found, time and space complexity.

**3.5.3.2.4 Interface with other functional component**

Not applicable

**3.5.3.2.5 Resource allocation**

style.css file.

**3.5.3.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3.3 Exponential search**

Teacher/ student

FALSE

TRUE

Fig 3.24 Exponential Search

**LEVEL 2**

**3.5.3.3.1 Input**

Array of numbers, key element

**3.5.3.3.1 Process**

It searches for an element in a sorted array by jumping 2^I elements every iteration where I represents the value of loop control variable, and then verifying if search element is present between last jump and current jump.

**3.5.3.3.3 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**3.5.3.3.4 Interface with other functional component**

Not applicable

**3.5.3.3.5 Resource allocation**

style.css file.

**3.5.3.3.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3.4 Interpolation search**

Teacher/ student

FALSE

TRUE

Fig 3.25 Interpolation Search

**LEVEL 2**

**3.5.3.4.1 Input**

Array of numbers, key element

**3.5.3.4.2 Process**

Interpolation search is an algorithm for searching for a key in an array that has been ordered by numerical values assigned to the keys (key values).

**3.5.3.4.3 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**3.5.3.4.4 Interface with other functional component**

Not applicable

**3.5.3.4.5 Resource allocation**

style.css file.

**3.5.3.4.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3.5 Jump search**

Teacher/ student

FALSE

TRUE

Fig 3.26 Jump Search

**LEVEL 2**

**3.5.3.5.1 Input**

Array of numbers, key element

**3.5.3.5.2 Process**

Jump search technique also works for ordered lists. It creates a block and tries to find the element in that block. If the item is not in the block, it shifts the entire block. The block size is based on the size of the list. If the size of the list is n then block size will be √n.

**3.5.3.5.3 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**3.5.3.5.4 Interface with other functional component**

Not applicable

**3.5.3.5.5 Resource allocation**

style.css file.

**3.5.3.5.6 User interface**

Buttons, labels, textbox, message box.

**3.5.3.6 Ternary search**

Teacher/ student

FALSE

TRUE

Fig 3.27 Ternary Search

**LEVEL 2**

**3.5.3.6.1 Input**

Array of numbers, key element

**3.5.3.6.2 Process**

Here the array elements are divided into 3 groups within the key element is searched.

**3.5.3.6.3 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**3.5.3.6.4 Interface with other functional component**

Not applicable

**3.5.3.6.5 Resource allocation**

style.css file.

**3.5.3.6.6 User interface**

Buttons, labels, textbox, message box.

**3.5.4 Stack**

Teacher/Staff

**LEVEL 1**

Fig 3.28 Stack Operation (Level 1)

**3.5.4.1 Array Implementation**

Teacher/Staff

**LEVEL 2**

Fig 3.29 Array Implementation

**3.5.4.1.1 Push**

**LEVEL 3**

Teacher/ student

FALSE

TRUE

Fig 3.30 Push Using Array Implementation

**3.5.4.1.1.1 Input**

Number

**3.5.4.1.1.2 Processing**

It stores the number at the beginning of the container and checks whether the stack is full.

**3.5.4.1.1.3 Output**

Enters the number within the container also calculates time and space complexity.

**3.5.4.1.1.4 Interface with other functional component**

Not applicable.

**3.5.4.1.1.5 Resource allocation**

style.css file.

**3.5.4.1.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.4.1.2 Pop**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.31 Pop Using Array Implementation

**3.5.4.1.2.1 Input**

Number.

**3.5.4.1.2.2 Processing**

It removes the last entered number and checks whether the stack empty.

**3.5.4.1.2.3 Output**

Removes the number from the container also calculates time and space complexity.

**3.5.4.1.2.4 Interface with other functional component**

Not applicable.

**3.5.4.1.2.5 Resource allocation**

style.css file.

**3.5.4.1.2. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.4.1.2 Linked list Implementation**

**LEVEL 2**

Teacher/Staff

Fig 3.32 Linked List Implementation

**3.5.4.2.1 Push**

**LEVEL 3**

Teacher/ student

FALSE

TRUE

Figs 3.33 Push Using Linked List

**3.5.4.2.1.1 Input**

Number.

**3.5.4.2.1.2 Processing**

It creates the node first and allocates memory if the list is empty then the node is pushed as the first element of the list otherwise link the previous element to the address field of new node.

**3.5.4.2.1.3 Output**

Pushes the element to the list also calculates time and space complexity.

**3.5.4.2.1.4 Interface with other functional component**

Not applicable.

**3.5.4.2.1.5 Resource allocation**

style.css file.

**3.5.4.2.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.4.2.2 Pop**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Figs 3.34 Pop Using Linked List

**3.5.4.2.2.1 Input**

Number.

**3.5.4.2.2.2 Processing**

Node is removed from the end of the linked list. Therefore, must delete the value stored in the head pointer. Then the following node will become the head node.

**3.5.4.2.2.3 Output**

Removes the element to the list also calculates time and space complexity.

**3.5.4.2.2.4 Interface with other functional component**

Not applicable.

**3.5.4.2.2.5 Resource allocation**

style.css file.

**3.5.4.2.2. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.5 Queue Module**

**LEVEL-1**

Teacher/Student

Fig 3.35 Queue Operation(Level-1)

**3.5.5.1 Array implementation**

**LEVEL 2**

Teacher/Staff

Fig 3.36 Queue Array Implementation

**3.5.5.1.1 Insertion**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.37 Insertion of Queue

**3.5.5.1.1.1 Input**

Number

**3.5.5.1.1 .2 Process**

It stores the number at the beginning of the container and checks whether the queue is full.

**3.5.5.1.1.3 Output**

Enters the number within the container also calculates time and space complexity.

**3.5.5.1.1.4 Interface with other functional component**

Not applicable.

**3.5.5.1.1.5 Resource allocation**

style.css file.

**3.5.5.1.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.5.1.2 Deletion**

Teacher/ student

TRUE

FALSE

**LEVEL-2**

Fig 3.38 Deletion of Queue

**3.5.5.1.2.1 Input**

Number

**3.5.5.1.2.2 Process**

It removes the first entered number and checks whether the queue empty.

**3.5.5.1.2.3 Output**

Removes the number from the container also calculates time and space complexity.

**3.5.5.1.2.4 Interface with other functional component**

Not applicable.

**3.5.5.1.2.5 Resource allocation**

style.css file.

**3.5.5.1.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.5.2 Linked List Implementation**

**LEVEL 2**

Teacher/Staff

Fig 3.39 Queue LinkedList Implementation

**3.5.5.2.1 Insertion**

Teacher/ Student

TRUE

FALSE

**LEVEL 3**

Fig 3.40 Queue Insertion

**3.5.5.2.1.1 Input**

Number

**3.5.5.2.1.2 Process**

It adds an element to the end of the queue. The new element becomes the last element of the queue.

**3.5.5.2.1.3 Output**

Enters the element to the list also calculates time and space complexity.

**3.5.5.2.1.4 Interface with other functional component**

Not applicable.

**3.5.5.2.1.5 Resource allocation**

style.css file.

**3.5.5.2.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.5.2.2 Deletion**

Teacher/ Student

FALSE

TRUE

Fig 3.41 Queue Deletion

**LEVEL-2**

**3.5.5.2.2.1 Input**

Number

**3.5.5.2.2.2 Process**

Removes the element which was first inserted in the queue.

**3.5.5.2.2.3 Output**

Removes the element from the list also calculates time and space complexity.

**3.5.5.2.2.4 Interface with other functional component**

Not applicable.

**3.5.5.2.2.5 Resource allocation**

style.css file.

**3.5.5.2.2. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.5.3 Circular Queue**

**LEVEL 2**

Teacher/Staff

Fig 3.42 Circular Queue

**3.5.5.3.1 Insertion**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.43 Circular Queue Insertion

**3.5.5.3.1.1 Input**

Number

**3.5.5.3.1.2 Process**

It stores the number at the beginning of the circle and checks whether the circular queue is full.

**3.5.5.3.1.3 Output**

Enters the number in the circle also calculates time and space complexity.

**3.5.5.3.1.4 Interface with other functional component**

Not applicable.

**3.5.5.3.1.5 Resource allocation**

style.css file.

**3.5.5.3.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.5.3.2 Deletion**

Teacher/ student

FALSE

TRUE

**LEVEL-2**

Fig 3.44 Circular Queue Deletion

**3.5.5.3.2.1 Input**

Number

**3.5.5.3.2.2 Process**

It removes the first entered number and checks whether the circular queue empty.

**3.5.5.3.2.3 Output**

Removes the number in the circle also calculates time and space complexity.

**3.5.4.1.1.4 Interface with other functional component**

Not applicable.

**3.5.4.1.1. 5 Resource allocation**

style.css file.

**3.5.4.1.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6 LinkedList**

**LEVEL 1**

Teacher/Student

Fig 3.45 Linked List(Level 1)

**3.5.6.1 Singly Linked list**

**LEVEL 2**

Teacher/Student

Fig 3.46 Singly LinkedList Operation

**3.5.6.1.1 Insertion at beginning**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.47 Insertion at Beginning

**3.5.6.1.1.1 Input**

Node elements

**3.5.6.1.1.2 Process**

Insert the node at the beginning of the Singly linked list.

**3.5.6.1.1.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.1.4 Interface with other functional component**

Not applicable.

**3.5.6.1.1. 5 Resource allocation**

style.css file.

**3.5.6.1.1. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.2 Insertion at end**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.48 Insertion at End

**3.5.6.1.2.1 Input**

Node elements

**3.5.6.1.2.2 Process**

Insert the node at the end of the Singly linked list.

**3.5.6.1.2.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.2.4 Interface with other functional component**

Not applicable.

**3.5.6.1.2.5 Resource allocation**

style.css file.

**3.5.6.1.2. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.3 Insertion at position**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.49 Insertion at Position

**3.5.6.1.3.1 Input**

Node elements

**3.5.6.1.3.2 Process**

Insert the node at the specific position of the Singly linked list.

**3.5.6.1.3.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.3.4 Interface with other functional component**

Not applicable.

**3.5.6.1.3.5 Resource allocation**

style.css file.

**3.5.6.1.3. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.4 Deletion at beginning**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.50 Deletion at Beginning

**3.5.6.1.4.1 Input**

Node elements

**3.5.6.1.4.2 Process**

Delete the node at the beginning of the Singly linked list.

**3.5.6.1.4.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.4.4 Interface with other functional component**

Not applicable.

**3.5.6.1.4.5 Resource allocation**

style.css file.

**3.5.6.1.4. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.5 Deletion at end**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.51 Deletion at The End

**3.5.6.1.5.1 Input**

Node elements

**3.5.6.1.5.2 Process**

Delete the node at the end of the Singly linked list.

**3.5.6.1.5.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.5.4 Interface with other functional component** Not applicable.

**3.5.6.1.5.5 Resource allocation**

style.css file.

**3.5.6.1.5. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.6 Deletion at position**

**LEVEL 3**

Teacher/ student

FALSE

TRUE

Fig 3.52 Deletion at Position

**3.5.6.1.6.1 Input**

Node elements

**3.5.6.1.6.2 Process**

Delete the node at specific location of the Singly linked list.

**3.5.6.1.6.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.6.4 Interface with other functional component**

Not applicable.

**3.5.6.1.6.5 Resource allocation**

style.css file.

**3.5.6.1.6.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.1.7 Deletion on element**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.53 Deletion on Element

**3.5.6.1.7.1 Input**

Element

**3.5.6.1.7.2 Process**

Delete the specific element of the Singly linked list.

**3.5.6.1.7.3 Output**

Displays the Singly linked list with time and space complexity.

**3.5.6.1.7.4 Interface with other functional component** Not applicable.

**3.5.6.1.7.5 Resource allocation**

style.css file.

**3.5.6.1.7. 6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2 Doubly Linked list**

**LEVEL 2**

Teacher/Student

Fig 3.54 Doubly Linked List

**3.5.6.2.1 Insertion at beginning**

**LEVEL 3**

Teacher/ student

TRUE

FALSE

Fig 3.55 Insertion at Beginning

**3.5.6.2.1.1 Input**

Node elements

**3.5.6.2.1.2 Process**

Insert the node at the beginning of the Doubly linked list.

**3.5.6.2.1.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.1.4 Interface with other functional component**

Not applicable.

**3.5.6.2.1.5 Resource allocation**

style.css file.

**3.5.6.2.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2.2 Insertion at end**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.56 Insertion at End

**3.5.6.2.2.1 Input**

Node elements

**3.5.6.2.2.2 Process**

Insert the node at the end of the Doubly linked list.

**3.5.6.2.2.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.2.4 Interface with other functional component**

Not applicable.

**3.5.6.2.2.5 Resource allocation**

style.css file.

**3.5.6.2.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2.3 Insertion at position**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.57 Insertion at Position

**3.5.6.2.3.1 Input**

Node elements

**3.5.6.2.3.2 Process**

Insert the node at the specific position of the Doubly linked list.

**3.5.6.2.3.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.3.4 Interface with other functional component**

Not applicable.

**3.5.6.2.3.5 Resource allocation**

style.css file.

**3.5.6.2.3.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2.4 Deletion at beginning**

**LEVEL 3**

Teacher/ student

TRUE

FALSE

Fig 3.58 Deletion at Beginning

**3.5.6.2.4.1 Input**

Node elements

**3.5.6.2.4.2 Process**

Delete the node at the beginning of the Doubly linked list.

**3.5.6.2.4.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.4.4 Interface with other functional component**

Not applicable.

**3.5.6.2.4. 5 Resource allocation**

style.css file.

**3.5.6.2.4.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2.5 Deletion at end**

**LEVEL 3**

Teacher/ student

TRUE

FALSE

Fig 3.59 Deletion at End

**3.5.6.2.5.1 Input**

Node elements

**3.5.6.2.5.2 Process**

Delete the node at the end of the Doubly linked list.

**3.5.6.2.5.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.5.4 Interface with other functional component**

Not applicable.

**3.5.6.2.5.5 Resource allocation**

style.css file.

**3.5.6.2.5.6 User interface**

Buttons, labels, textbox, message box.

* + - * 1. **Deletion at position**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.60 Deletion at Position

**3.5.6.2.6.1 Input**

Node elements

**3.5.6.2.6.2 Process**

Delete the node at specific location of the Doubly linked list.

**3.5.6.2.6.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.6.4 Interface with other functional component**

Not applicable.

**3.5.6.2.6. 5 Resource allocation**

style.css file.

**3.5.6.2.6.6 User interface**

Buttons, labels, textbox, message box.

**3.5.6.2.7 Deletion at element**

**LEVEL 3**

Teacher/ student

TRUE

FALSE

Fig 3.61 Deletion at Element

**3.5.6.2.7.1 Input**

Node elements

**3.5.6.2.7.2 Process**

Delete the specific element of the Doubly linked list.

**3.5.6.2.7.3 Output**

Displays the Doubly linked list with time and space complexity.

**3.5.6.2.7.4 Interface with other functional component**

Not applicable.

**3.5.6.2.7. 5 Resource allocation**

style.css file.

**3.5.6.2.7.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7 Tree**

**LEVEL 1**

Teachers /student

Fig 3.62 Tree (Level 1)

**3.5.7.1 Binary Search Tree**

**LEVEL 2**

Teacher/Student

Fig 3.63 Binary Search Tree

**3.5.7.1.1 Insertion**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.64 Insertion

**3.5.7.1.1.1 Input**

Number

**3.5.7.1.1.2 Process**

This technique is used to add new element to the binary search tree in appropriate location.

**3.5.7.1.1.3 Output**

Displays the binary tree including the new element also calculates the time and space complexity.

**3.5.7.1.1.4 Interface with other functional component**

Not applicable.

**3.5.7.1.1.5 Resource allocation**

style.css file.

**3.5.7.1.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7.1.2 Deletion**

**LEVEL 3**

Teacher/ student

FALSE

TRUE

Fig 3.65 Deletion

**3.5.7.1.2.1 Input**

Number

**3.5.7.1.2.2 Process**

This technique will delete the specified from a binary search tree.

**3.5.7.1.2.3 Output**

Displays the binary tree excluding the removed element also calculates the time and space complexity.

**3.5.7.1.2.4 Interface with other functional component**

Not applicable.

**3.5.7.1.2.5 Resource allocation**

style.css file.

**3.5.7.1.2.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7.1.3 Searching**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.66 Searching

**3.5.7.1.3.1 Input**

key element

**3.5.7.1.3.2 Process**

This technique is used for locating a key from within a set.

**3.5.7.1.3.3 Output**

Displays the binary tree with searched element position also calculates the time and space complexity.

**3.5.7.1.3.4 Interface with other functional component**

Not applicable.

**3.5.7.1.3.5 Resource allocation**

style.css file.

**3.5.7.1.3.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7.1.4 Pre-order**

Teacher/ student

FALSE

TRUE

**LEVEL 3**

Fig 3.67 Pre-Order

**Input**

List of numbers.

**3.5.7.1.4.2 Process**

This technique that traverses from root to the left subtree then to the right sub tree.

**3.5.7.1.4.3 Output**

Displays the list of traversed order.

**3.5.7.1.4.4 Interface with other functional component**

Not applicable.

**3.5.7.1.4.5 Resource allocation**

style.css file.

**3.5.7.1.4.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7.1.5 Post-order**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.68 Post-Order

**3.5.7.1.5.1 Input**

List of numbers.

**3.5.7.1.5.2 Process**

This technique that traverses from left subtree to the right subtree then to the root.

**3.5.7.1.5.3 Output**

Displays the list of traversed order.

**3.5.7.1.5.4 Interface with other functional component**

Not applicable.

**3.5.7.1.5.5 Resource allocation**

style.css file.

**3.5.7.1.5.6 User interface**

Buttons, labels, textbox, message box.

**3.5.7.1.6 In-order**

Teacher/ student

TRUE

FALSE

**LEVEL 3**

Fig 3.69 In-Order

**3.5.7.1.6.1 Input**

List of numbers.

**3.5.7.1.6.2 Process**

This technique that traverses from left subtree to the root then to the right subtree.

**3.5.7.1.6.3 Output**

Displays the list of traversed order.

**3.5.7.1.6.4 Interface with other functional component**

Not applicable.

**3.5.7.1.6.5 Resource allocation**

style.css file.

**3.5.7.1.6.6 User interface**

Buttons, labels, textbox, message box.

**3.5.8 Graph**

Teacher/Staff

Fig 3.70 Graph (Level 1)

**LEVEL 1**

**3.5.8.1 BFS**

Teacher/ student

TRUE

FALSE

**LEVEL 2**

Fig 3.71 BFS

**3.5.8.1.1 Input**

Enter Source and Destination Node.

**3.5.8.1.2 Process**

BFS or Breadth-First-Search starts from the top node in the graph and travels down until it reached the root node.

**3.5.8.1.3 Output**

It will display the complete from Source to Destination Node in given tree.

**3.5.8.1.4 Interface with other functional component**

Not applicable.

**3.5.8.1.5 Resource allocation**

style.css file.

**3.5.8.1.6 User interface**

Buttons, labels, textbox, message box.

**3.5.8.2 DFS**

Teacher/ student

TRUE

FALSE

**LEVEL 2**

Fig 3.72 DFS

**3.5.8.2.1 Input**

Enter Source and Destination Node.

**3.5.8.2.2 Process**

DFS or Depth-First-Search starts from the top node and follows the path to reaches the end node of the path.

**3.5.8.2.3 Output**

It will display the complete from Source to Destination Node in given tree

**3.5.8.2.4 Interface with other functional component**

Not applicable.

**3.5.8.2.5 Resource allocation**

style.css file.

**3.5.8.2.6 User interface**

Buttons, labels, textbox, message box.

**4. DETAILED DESIGN**

**4.1 Introduction**

During detailed design, the internal logic of each module specified in system design is decided. During this phase further details of the modules are decided. Design of each of the modules usually specified in a high-level description language which is independent of the language in which software eventually be implemented.

**4.2 Structure of software package**

**PERFORMANCE ANALYSIS AND SIMULATION OF DATA STRUCTURE ALGORITHMS**

A

B

C

E

F

G

H

Sorting

Array Operation

Searching

Stack

Queue

Linked List

Tree

Graph

D

A

Bubble Sort

Selection Sort

Insertion Sort

Exchange Sort

Counting Sort

Heap Sort

Bucket Sort

Merge Sort

Quick Sort

Brick Sort

Shell Sort

B

Insertion

Deletion

C

Linear Search

Binary Search

Exponential Search

Interpolation Search

Jump Search

Ternary Search

D

Array implementation

Push

Pop

Linked list Implementation

Push

Pop

E

Array implementation

Insertion

Deletion

Linked list Implementation

Insertion

Deletion

F

Singly Linked List

Doubly Linked List

1

2

1

Insertion at begin

Insertion at end

Insertion at position

Deletion at begin

Deletion at end

Deletion at position

2

Insertion at begin

Insertion at end

Insertion at position

Deletion at begin

Deletion at end

Deletion at position

G

Binary Search Tree

Insertion

Deletion

Searching

Pre-order

Post-order

In-order

H

BFS

DFS

Fig 4.1 Structure of Software Package

**4.3 Module decomposition of software**

**Structure chart:**

Structure chart is a top-down modular design, consist of squares representing different modules in a system and lines. Structure chart shows how program has been partitioned into manageable modules hierarchy and organization of those modules and communicational interface.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Process** |
|  | Data flow | Show the direction flow of data. |
|  | Control flow | Shows the direction of flow control. |
|  | Processing | Shows manipulation, calculation and processing. |
|  | Module Invocation | It represents subordinate module being invoked by superior ordinate module. |
| C  B  A  Main | Condition invocation | It indicates that the invocation of subordinate modules depends on the evaluation of a condition. |
| B  A  Main | Iteration | It represents the iteration. |

**Table 4.1 Structure chart**

**Flow chart:**

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Purpose** |
|  | Terminator | It indicates the start and end process. |
|  | Input/output | Input/output data. |
|  | Decision | It represents a comparison or question that determines an alternate path to be followed. |
|  | Flow direction | Shows the direction of the data flow. |
|  | Processing | It represents manipulation, calculation or information processing |
|  | Direction access storage. | File storage. |
|  | Preparation (Looping) | An instruction or group of instruction. |
|  | In-page |  |
|  | Off-page |  |
|  | Delay |  |
|  | Pre-defined Process |  |

Flow chart is a graphical representation of solution to the given problems. A flow chart is pictorial representation of an algorithm, workflow or process. The diagrammatic representation illustrates a solution model to given problem. It uses the following symbols.

**Table 4.2 Flow chart**

**4.3.1 Sorting module**

**4.3.1.1 Bubble sort**

**4.3.1.1.1 Inputs**

Array of numbers.

**4.3.1.1.2 Procedural details**

**Flowchart:**

NO

YES

NO

Start

Input array elements

Validation

Display error message

for I=1 to N-1

for J=1 to N-1

If data[J]>

data[J+1]

temp=data[J], data[J]=data[J+1]

data[J+1] =temp

YES

Visualizing of bubble sort algorithm

Time complexity

Best=Ω(n)

average=θ(n^2)

worst=O(n^2)

Space complexity

Worst=O (1)

Display time and space complexity

End

Fig 4.2 Bubble sort

**4.3.1.1.3 File Input/Output Interface**

Style.CSS

**4.3.1.1.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.1.5 Implementation aspects**

Buttons, Textbox, Labels

**4.3.1.2 Selection sort**

**4.3.1.2.1 Inputs**

Array of numbers

**4.3.1.2.2 Procedural details**

**Flowchart:**

NO

YES

NO

Start

Input array elements

Validation

Display error message

for I=1 to N-1

If A[J]<

Small

Small=A[J], pos=J

YES

Small=A[I], pos=I

for J=I+1 to N

A[pos]=A[I], A[I]=Small

Visualizing of selection sort algorithm

Time complexity

Best=Ω(n^2)

average=θ(n^2)

worst=O(n^2)

Space complexity

Worst=O (1)

Display time and space complexity

End

Fig 4.3 Selection sort

**4.3.1.2.3 File Input/Output Interface**

Style.CSS

**4.3.1.2.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.2.5 Implementation aspects**

Buttons, labels, Textbox

**4.3.1.3 Insertion sort**

**4.3.1.3.1 Input**

Array of numbers

**4.3.1.3.2 Procedural details**

**Flowchart:**

NO

YES

NO

Start

Input array elements

Validation

Display error message

for I=1 to N

If A[J]>

temp

A[J+1] =A[J]

temp=A[I]

for J=I-1 to 1 by -1

A[J+1] =temp

YES

Visualizing of insertion sort algorithm

Time complexity

Best=Ω(n)

average=θ(n^2)

worst=O(n^2)

Space complexity

Worst=O (1)

Display time and space complexity

End

Fig 4.4 Insertion sort

**4.3.1.3.3 File Input/Output Interface**

Style.CSS

**4.3.1.3.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.3.5 Implementation aspects**

Buttons, labels, Textbox

**4.3.1.4 Exchange sort**

**4.3.1.4.1 Input**

Array of numbers

**4.3.1.4.2 Procedural details**

**Flowchart:**

Visualizing of exchange sort algorithm

Time complexity

Best=Ω(n)

average=θ(n^2)

worst=O(n^2)

Space complexity

Worst=O (1)

Display time and space complexity

End

NO

YES

NO

Start

Input array elements

Validation

Display error message

for I=1 to N-2

If Num[I]>

Num[J]

temp

Swap (Num[I], Num[J])

for J=I+1 to N-1

YES

Fig 4.5 Exchange sort

**4.3.1.4.3 File Input/Output Interface**

Style.CSS

**4.3.1.4.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.4.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.5 Counting sort**

**4.3.1.5.1 Input**

Array of numbers

**4.3.1.5.2 Procedural details**

**Flowchart:**

Visualizing of counting sort algorithm

YES

NO

yes

If a[i]=j

for i=1 to n

max=max(a)

NO

Start

Input array elements

Validation

Display error message

Count[i]+=1

for i=0 to max

Count[i]=0

for j=0 to max

Time complexity

Best=Ω (n + k)

average=θ (n + k)

worst=O (n + k)

Space complexity

Worst=O (k)

Display time and space complexity

End

Fig 4.6 Counting sort

**4.3.1.5.3 File Input/Output Interface**

Style.CSS

**4.3.1.5.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.5.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.6 Heap sort**

**4.3.1.6.1 Inputs**

Array of numbers

**4.3.1.6.2 Procedural detail**

**Flowchart:**

Visualizing of heapsort algorithm

Time complexity

Best=Ω (n log(n))

average=θ (n log(n))

worst=O (n ^ 2)

Space complexity

Worst=O (n)

Display time and space complexity

End

Start

Input array

elements

validation

NO

YES

Heapsort (arr)

Heapsort (arr)

For i=length(arr) to 2

Swap(arr[1], arr[i]), heap\_size[arr]=heap\_size[arr]+1

\_

Maxheapify(arr,1)

Maxheapify (arr, i)

l=left(i), r=right(i)

If i>heap\_size[arr]&&arr[l]>arr[i]

Largest=l

Largest=i

If i>heap\_size[arr]&&arr[l]>arr[i]

largest=r

If largest! =i

Swap(arr[i],arr[largest])

Maxheapify(arr, largest)

YES

NO

YES

NO

YES

NO

Buildmaxheap(arr)

Heap\_size(arr)=length(arr)

For i=length(arr)/2 to 1

Maxheap(arr,i)

Fig 4.7 Heap sort

**4.3.1.6.3 File Input/Output Interface**

Style.CSS

**4.3.1.6.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.6.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.7 Bucket sort**

**4.3.1.7.1 Input**

Array of numbers

**4.3.1.7.2 Procedural details**

**Flowchart:**

Bucket(a[i]) ++

YES

for i=1 to n

max=max(a)

NO

Start

Input array elements

Validation

Display error message

a[j++]=i, bucket[i]--

for i=0 to max

bucket[i]=0

for i=0, j=0 to max

While bucket[i]>0

TRUE

FALSE

Visualizing of bucket sort algorithm

Time complexity

Best=Ω (n + k)

average=θ (n + k)

worst=O (n ^ 2)

Space complexity

Worst=O (n)

Display time and space complexity

End

Fig 4.8 Bucket sort

**4.3.1.7.3 File Input/Output Interface**

Style.CSS

**4.3.1.7.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.7.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.8 Merge sort**

**4.3.1.8.1 Input**

Array of numbers

**4.3.1.8.2 Procedural details**

**Flowchart:**

i=low, j=mid+1, k=low

NO

Start

Input array elements

Validation

Display error message

If a[i]>a[j]

C[k]=a[j], j=j+1, k=k+1

C[k]=a[i], i=i+1, k=k+1

While (i<=mid) &&(j<=high)

While (i<=mid)

C[k]=a[i], i=i+1, k=k+1

While (j>=high)

C[k]=a[j], j=j+1, k=k+1

YES

TRUE

FALSE

YES

NO

YES

FALSE

TRUE

FALSE

TRUE

For i=low to high

A[i]=c[i]

Visualizing of merge sort algorithm

Time complexity

Best=Ω (n log(n))

average=θ (n log(n))

worst=O (n log(n))

Space complexity

Worst=O (n)

Display time and space complexity

End

Fig 4.9 Merge sort

**4.3.1.8.3 File Input/Output Interface**

Style.CSS

**4.3.1.8.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.8.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.9 Quick sort**

**4.3.1.9.1 Input**

Array of numbers

**4.3.1.9.2 Procedural details**

**Flowchart:**

Start

Input array

elements

validation

Quicksort (array a, start, end)

Visualizing of quicksort algorithm

Time complexity

Best=Ω (n log(n))

average=θ (n log(n))

worst=O (n ^ 2)

Space complexity

Worst=O (n)

Display time and space complexity

End

YES

NO

Quicksort (array a, start, end)

If start>end

P=partial (A, start, end)

YES

Quicksort (A, start, p-1)

Quicksort (A, p+1, end)

NO

Partition (Array a, start, end)

Pivot=A[end], i=start-1

For j=start to end-1

If(A[j]<pivot

i=i+1, swap(A[i+1], a[j]

Swap(A[j+1], A[end]

YES

NO

Fig 4.10 Quick sort

**4.3.1.9.3 File Input/Output Interface**

Style.CSS

**4.3.1.9.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.9.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.10 Brick sort**

**4.3.1.10.1 Input**

Array of numbers

**4.3.1.10.2 Procedural details**

YES

YES

NO

YES

If list[i]>list[i+1]

for i=1 to n by 2

NO

Start

Input array elements

Validation

Display error message

Temp=list[i], list[i]=list[i+1], list[i]=temp, sorted=false

Sorted=true

While (! Sorted)

FALSE

TRUE

**Flowchart:**

for i=0 to n-1 by 2

If list[i]>list[i+1]

Temp=list[i], list[i]=list[i+1], list[i]=temp, sorted=false

Visualizing of brick sort algorithm

Time complexity

Best= θ(n)

average= θ(n\*n)

worst= θ(n\*n)

worst=O (n log(n))

Space complexity

Worst= θ (1)

Display time and space complexity

End

NO

YES

Fig 4.11 Brick sort

**4.3.1.10.3 File Input/Output Interface**

Style.CSS

**4.3.1.10.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.10.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.1.11 Shell sort**

**4.3.1.11.1 Input**

Array of numbers

**4.3.1.11.2 Procedural details**

Start

Input array elements

NO

Validation

Display error message

Gap=(interval\*3) +1

While(gap<length(array)/3)

**Flowchart:**

YES

Visualizing of shell sort algorithm

Time complexity

Best=Ω (n log (n))

average=θ (n log (n))

worst=O (n ^ 2)

Space complexity

Worst=O (n)

Display time and space complexity

End

Array[inner]=array[inner-gap], inner=inner-gap

Insertion\_val=array[outer], inner=outer

For outer=gap;outer<length(array);outer++

While(gap>0)

While(innere>gap-1 && array[inner-gap]>=insertion\_val

TRUE

FALSE

TRUE

FALSE

gap=(gap-1)/3

Array[inner]=insertion\_val

outer

Fig 4.12 Shell sort

**4.3.1.11.3 File Input/Output Interface**

Style.CSS

**4.3.1.11.4 Output**

Displays the sorted bar graph also calculates the time and space complexity.

**4.3.1.11.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.2 Array Operation**

**4.3.2.1 Insertion**

**4.3.2.1.1 Inputs**

Array of numbers, key element, position

**4.3.2.1.2 Procedural details**

**Algorithm:**

**INSERT (LA, N, K, ITEM)**

Step 1: Start.

Step 2: Input array element, insert, key element.

Step 3: IF invalid input THEN

goto step 2

ELSE

* + - * 1. [Initialize counter] SET J: =N
        2. REPEAT WHILE J>=K

1. [ Move Jth element downward] SET LA[J+1]: = LA[J]
2. [Decrease counter] SET J: = J-1

[END OF WHILE LOOP]

* + - * 1. [Insert element] SET LA[K]: = ITEM
        2. [Reset N] SET N: =N+1
        3. Calculate time complexity SET BEST=O(N), AVERAGE=O(N2), WORST=(N2)
        4. Calculate space complexity SET SPACE=O(1)
        5. Visualization of array insertion.
        6. Display result.

[END OF IF]

Step 7: Exit.

**4.3.2.1.3 File Input/Output Interface**

Style.CSS

**4.3.2.1.4 Output**

Displays the array with inserted element using bar graph also calculates the time and space complexity.

**4.3.2.1.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.2.2 Deletion**

**4.3.2.2.1 Inputs**

Array of numbers, number to be deleted.

**4.3.2.2.2 Procedural details**

**Algorithm:**

**DELETE (LA, N, K, ITEM)**

Step 1: Start

Step 2: Input the number to be deleted.

Step 3: IF invalid input THEN

goto step 2.

ELSE

* + - * 1. SET ITEM = LA[K]
        2. REPEAT FOR J=K TO N-1

1. [Move J+1st element upward] SET LA[J] = LA[J+1]

[END OF LOOP]

* + - * 1. [Reset the number N of elements in LA] SET N: = N-1
        2. Calculate time complexity SET BEST=O(N), AVERAGE=O(N2), WORST=(N2)
        3. Calculate space complexity SET SPACE=O(1)
        4. Visualization of array insertion.
        5. Display result.

[END OF IF]

Step 4: Exit

**4.3.2.2.3 File Input/Output Interface**

Style.CSS

**4.3.2.2.4** **Output**

Displays the final bar graph excluding the deleted element and then displays the position of the element deleted also calculates the time and space complexity.

**4.3.2.2.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3 Searching**

**4.3.3.1 Linear Search**

**4.3.3.1.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.1.2 Procedural details**

**Structure chart:**

Linear search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

Fig 4.13 Linear Search

* + - * 1. **File Input/Output Interface**

Style.CSS

**4.3.3.1.4 Output**

Displays the searched element in the bar graph with different color and also displays the position of the element found, time and space complexity.

**4.3.3.1.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3.2 Binary Search**

**4.3.3.2.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.2.2 Procedural details**

**Structure chart:**

Binary search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

Fig 4.14 Binary Search

* + - * 1. **File Input/Output Interface**

Style.CSS

**4.3.3.2.4** **Output**

Displays the searched element in the bar graph with different color and also displays the position of the element found, time and space complexity.

**4.3.3.2.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3.3 Exponential Search**

**4.3.3.3.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.3.2 Procedural details**

**Structure chart:**

Exponential search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

Fig 4.15 Exponential Search

**4.3.3.3.3 File Input/Output Interface**

Style.CSS

**4.3.3.3.4 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**4.3.3.3.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3.4 Interpolation Search**

**4.3.3.4.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.4.2 Procedural details**

Interpolation search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

**Structure chart:**

Fig 4.16 Interpolation Search

* + - * 1. **File Input/Output Interface**

Style.CSS

**4.3.3.4.4** **Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**4.3.3.4.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3.5 Jump Search**

**4.3.3.5.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.5.2 Procedural details**

**Structure chart:**

Jump search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

Fig 4.17 Jump Search

* + - * 1. **File Input/Output Interface**

Style.CSS

**4.3.3.5.4 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**4.3.3.5.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.3.6 Ternary Search**

**4.3.3.6.1 Inputs**

Array of numbers, number to be searched.

**4.3.3.6.2 Procedural details**

**Structure chart:**

Ternary search

Input details

Array of numbers, number to be searched

Validation

Search the element

Display the result

Calculate time and space complexity

Fig 4.18 Ternary Search

* + - * 1. **File Input/Output Interface**

Style.CSS

**4.3.3.6.4 Output**

Displays the searched element along the sorted bar graph also the time and space complexity.

**4.3.3.6.5 Implementation aspects**

Buttons, Labels, Textbox

**4.3.4 Stack**

**4.3.4.1 Array Implementation**

**4.3.4.1.1 Push**

**4.3.4.1.1.1 Input**

Numbers

**4.3.4.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

Goto step 2

ELSE

1. [Stack already filled?]

IF TOP = MAXSTK, THEN

Print: “Stack overflow" and RETURN

[END OF IF]

1. SET TOP = TOP + 1
2. STACK[TOP]=ITEM
3. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
4. Calculate space complexity SET SPACE=O (1)
5. Visualization of stack push operation.
6. Display result.

[END OF IF]

Step 4:Exit

**4.3.4.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.4.1.1.4 Output**

Enters the number within the container also calculates time and space complexity.

**4.3.4.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.4.1.2 Pop**

**4.3.4.1.2.1 Input**

Numbers

**4.3.4.1.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

Goto step 2

ELSE

1. [Stack has item to be removed]

IF TOP = MAXSTK THEN

Print: “Stack underflow" and RETURN

[END OF IF]

1. SET ITEM = STACK[TOP]
2. SET TOP = TOP – 1
3. Calculate time complexity SET BEST=O (1), AVERAGE=O(1), WORST=O(N)
4. Calculate space complexity SET SPACE =O (1)
5. Visualization of pop operation.
6. Display result.

[END OF IF]

Step 4: Exit

**4.3.4.1.2.3 File Input/Output Interfaces**

Style.css

**4.3.4.1.2.4 Output**

Removes the number from the container also calculates time and space complexity.

**4.3.4.1.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.4.2 Linked List Implementation**

**4.3.4.2.1 Push**

**4.3.4.2.1.1 Input**

Numbers

**4.3.4.2.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

Goto step 2

ELSE

1. Create a new NODE SET NEWNODE = new NODE
2. INPUT ITEM
3. SET NEWNODE -> DATA = ITEM
4. IF TOP = NULL THEN
5. SET TOP = NEWNODE
6. NEWNODE -> LINK = NULL

ELSE

1. SET NEWNODE -> LINK = TOP
2. SET TOP = NEWNODE

[END OF IF]

1. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
2. Calculate space complexity SET SPACE=O(1)
3. Visualization of push operation.
4. Display result.

[END OF IF]

Step 5: Exit

**4.3.4.2.1.3 File Input/Output Interfaces**

Style.css

**4.3.4.2.1.4 Output**

Pushes the element to the list also calculates time and space complexity.

**4.3.4.2.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.4.2.2 Pop**

**4.3.4.2.2.1 Input**

Numbers

**4.3.4.2.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF( TOP = NULL) THEN

WRITE: "stack is empty"

ELSE

1. SET TEMP = TOP
2. SET TOP = TOP -> LINK
3. WRITE: TEMP -> DATA
4. DELETE (TEMP)

[END OF IF]

1. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
2. Calculate space complexity SET SPACE=O (1)
3. Visualization of pop operation of stack.
4. Display result.

[END OF IF]

Step 4: Exit

**4.3.4.2.2.3 File Input/Output Interfaces**

Style.css

**4.3.4.2.2.4 Output**

Removes the element to the list also calculates time and space c complexity.

**4.3.4.2.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5 Queue**

**4.3.5.1 Array Implementation**

**4.3.5.1.1 Insertion**

**4.3.5.1.1.1 Input**

Numbers

**4.3.5.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. [Queue already filled?]

IF[REAR==MAXSIZE) THEN

WRITE:” Overflow” and RETURN

[END OF IF]

1. SET REAR: =REAR+1
2. QUEUE[REAR]: =ITEM
3. IF(FRONT==0) THEN

SET FRONT: =1

[END OF IF]

1. Calculate time complexity best=O (1), average=O (1), worst=O(n)
2. Calculate space complexity SET SPACE=O (1)
3. Visualization of insertion operation.
4. Display result.

[END OF IF]

Step 5: Exit

**4.3.5.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.5.1.1.4 Output**

Enters the number within the container also calculates time and space complexity.

**4.3.4.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5.1.2 Deletion**

**4.3.5.1.2.1 Input**

Numbers

**4.3.5.1.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. [Check queue is empty?]

IF[FRONT==0) THEN

WRITE:” Underflow” and RETURN

[END OF IF]

1. SET ITEM: =QUEUE[FRONT]
2. IF(FRONT=REAR) THEN
3. SET FRONT: =1
4. SET FRONT: =0

ELSE

1. SET FRONT: =FRONT+1

[END OF IF]

1. Calculate time complexity best=O (1), average=O (1), worst=O(n)
2. Calculate space complexity SET SPACE=O(1)
3. Visualization of deletion operation.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.5.1.2.3 File Input/Output Interfaces**

Style.css

**4.3.5.1.2.4 Output**

Removes the number from the container also calculates time and space complexity.

**4.3.5.1.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5.2 Linked List Implementation**

**4.3.5.2.1 Insertion**

**4.3.5.2.1.1 Input**

Numbers

**4.3.5.2.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

* + - * 1. Create a new NODE SET NEWNODE = new NODE
        2. INPUT ITEM
        3. SET NEWNODE -> DATA = ITEM
        4. SET NEWNODE -> LINK = NULL
        5. IF(FRONT=NULL) THEN

1. SET FRONT: =NEWNODE
2. SET REAR: =NEWNODE

ELSE

SET REAR->LINK: =NEWNODE

SET REAR: =NEWNODE

[END OF IF]

* + - * 1. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
        2. Calculate space complexity SET SPACE=O (1)
        3. Visualization of insertion operation.
        4. Display result

[END OF IF]

Step 4: Exit

**4.3.5.2.1.3 File Input/Output Interfaces**

Style.css

**4.3.5.2.1.4 Output**

Enters the element to the list also calculates time and space complexity.

**4.3.5.2.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5.2.2 Deletion**

**4.3.5.2.2.1 Input**

Numbers

**4.3.5.2.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

Goto step 2

ELSE

1. IF (FRONT = NULL) THEN

WRITE: "queue is empty"

ELSE

* + - * 1. SET ITEM: =FRONT->DATA
        2. SET FRONT: =FRONT->LINK
        3. WRITE: “DELETE THE ITEM” ITEM

[END OF IF]

1. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
2. Calculate space complexity SET SPACE=O (1)
3. Visualization of deletion operation.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.5.2.2.3 File Input/Output Interfaces**

Style.css

**4.3.5.2.2.4 Output**

Removes the element from the list also calculates time and space complexity.

**4.3.5.2.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5 Circular Queue**

**4.3.5.3.1 Insertion**

**4.3.5.3.1.1 Input**

Numbers

**4.3.5.3.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF (FRONT: =(REAR+1) %MAXSIZE) THEN

WRITE: ”Cqueue is full” and RETURN

[END OF IF]

1. SET REAR: =(REAR+1) %MAXSIZE
2. SET CQUEUE[REAR]: =ITEM
3. IF(FRONT==-1) THEN

SET FRONT: =0

[END OF IF]

1. Calculate time complexity SET BEST=O (1), AVERAGE=O(N), WORST=O (1)
2. Calculate space complexity SET SPACE=O(n)
3. Visualization of insertion operation of circular queue.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.5.3.1.3 File Input/Output Interfaces**

Style.css

**4.3.5.3.1.4 Output**

Enters the number in the circle also calculates time and space complexity.

**4.3.4.3.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.5.3.2 Deletion**

**4.3.5.3.2.1 Input**

Numbers

**4.3.5.3.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF[FRONT=-1) THEN

WRITE:”Element is empty” and RETURN

[END OF IF]

1. SET ITEM: =CQUEUE[FRONT]

IF(FRONT=REAR) THEN

1. SET FRONT: =-1
2. SET FRONT: =-1

ELSE

SET FRONT: =(FRONT+1) %MAXSIZE

[END OF IF]

1. Calculate time complexity SET BEST=O (1), AVERAGE=O (1), WORST=O(N)
2. Calculate space complexity SET SPACE=O (1)
3. Visualization of deletion operation of circular queue.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.5.3.2.3 File Input/Output Interfaces**

Style.css

**4.3.5.3.2.4 Output**

Removes the number in the circle also calculates time and space complexity.

**4.3.5.3.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6 Linked List**

**4.3.6.1 Singly linked list**

**4.3.6.1.1 Insertion at beginning**

**4.3.6.1.1.1 Input**

Array of numbers

**4.3.6.1.1.2 Procedural detail**

**Algorithm:**

Step 1: Start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. CREATE a NODE SET newnode: = new NODE;
2. SET newnode->data=element;
3. IF (head==NULL) THEN
4. SET head: =newnode;
5. SET newnode->link: =NULL;

ELSE

* + - * 1. SET newnode->link: =head;
        2. SET head: =newnode;

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O (1)
2. Calculate space complexity SET SPACE=O(2n+n+2)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.6.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.1.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.2 Insertion at end**

**4.3.6.1.2.1 Input**

Array of numbers

**4.3.6.1.2.2 Procedural detail**

**Algorithm:**

Step 1: Start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. Create a NODE SET newnode:= new NODE;
2. SET newnode->data=element;
3. SET newnode->link:=NULL
4. IF(head==NULL)THEN

SET head:=newnode;

ELSE

1. SET temp:=head;
2. REPEAT STEP

WHILE (temp->link! =NULL)

1. SET temp: =temp->link;

[END OF WHILE]

1. SET temp->link: =newnode;

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O(n1/2\*n)
2. Calculate space complexity SET SPACE=O(2n+n+2)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 5: Exit

**4.3.6.1.2.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.2.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.3 Insertion at position**

**4.3.6.1.2.1 Input**

Array of numbers

**4.3.6.1.2.2 Procedural detail**

**Algorithm:**

Step 1: Start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. CREATE a NODE SET newnode: = new NODE;
2. SET newnode->data=element;
3. SET temp: =head;
4. IF((pos=0) &&(head==NULL)) THEN
5. SET newnode->link: =NULL;
6. SET head: =newnode;

ELSE IF((pos=0) && (head! =NULL))

1. newnode->link=head;
2. head: =newnode;

ELSE

1. REPEAT STEP FOR i=0 TO i<pos-1

SET temp: =temp->link;

[END OF FOR]

1. IF(temp==NULL)THEN

WRITE: “Node in the list are less than position”

ELSE

1. SET newnode->link:=temp->link;
2. SET temp->link:=newnode;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(2n+n+4)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 5: Exit

**4.3.6.1.2.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.2.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.4 Deletion at beginning**

**4.3.6.1.1.1 Input**

Array of numbers

**4.3.6.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF(head==NULL)THEN

WRITE: “Linked list is empty”

ELSE

1. SET temp:= head
2. SET head=head->link;
3. SET ITEM:=temp->data
4. WRITE: ITEM
5. Delete(temp);

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O (1)
2. Calculate space complexity SET SPACE=O(2n+n+2)
3. Visualization of linked list.
4. Display result

[End of If]

Step 4: Exit

**4.3.6.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.1.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.5 Deletion at end**

**4.3.6.1.1.1 Input**

Array of numbers

**4.3.6.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF (head==NULL) THEN

WRITE: “Linked list is empty”

ELSE

1. SET temp1: =NULL
2. SET temp2: =head;
3. REPEAT STEP i & ii

WHILE (temp2->link! =NULL)

1. SET temp1: =temp2;
2. SET teemp2: =temp2->link;

[END OF WHILE]

1. WRITE: “deleted element is:” temp2->data;
2. Delete(temp2);
3. IF (temp1! =NULL) THEN

SET temp1->link: =NULL;

ELSE

SET head: =NULL;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY= O(n1/2\*n)
2. Calculate space complexity SET SPACE=O(2n+n+2)
3. Visualization of insertion operation of circular queue.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.6.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.1.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.6 Deletion at specified position**

**4.3.6.1.1.1 Input**

Array of numbers

**4.3.6.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF(head==NULL)THEN

WRITE: “Linked list is empty”

ELSE

1. SET temp1: =NULL
2. SET temp 2: =head;
3. REPEAT STEP i & ii

WHILE (temp2->link! =NULL)

(i)SET temp1: =temp2;

(ii) SET teemp2: =temp2->link;

[END OF WHILE]

1. WRITE: “deleted element is:” temp2->data;
2. Delete(temp2);
3. IF (temp1! =NULL) THEN

SET temp1->link: =NULL;

ELSE

SET head: =NULL;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(2n+n+4)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.6.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.6.1.1.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.6.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.1.7 Deletion on element**

**4.3.7.1.1.1 Input**

Array of numbers

**4.3.7.1.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input element

Step 3: IF not validate THEN

goto step 2

ELSE

1. SET START: = FIRST NODE
2. IF (START== NULL) THEN

WRITE: “list is empty”

[END OF IF]

1. WHILE (START! =NULL)

IF(START==ELEMENT) THEN

1. Delete(START)
2. SET START:= NEXT NODE

[END OF IF]

[END OF WHILE]

1. Calculate time complexity SET TIME\_COMPLEXITY=O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(2n+n+4)
3. Visualization of Singly linked list.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.7.1.1.3 File Input/Output Interfaces**

Style.css

**4.3.7.1.1.4 Output**

Displays the Singly linked list with time and space complexity.

**4.3.7.1.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2 Doubly linked list**

**4.3.6.2.1 Insertion at beginning**

**4.3.6.2.1.1 Input**

Array of numbers

**4.3.6.2.1.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. CREATE A NEW NODE SET NEWNODE:=new NODE
2. INPUT ITEM
3. SET NEWNODE->DATA:=ITEM
4. SET NEWNODE->LEFT:=NULL
5. SET NEWNODE->RIGHT:=NULL
6. IF(HEAD!=NULL)THEN
7. SET NEWNODE->RIGHT:=HEAD
8. SET HEAD->LEFT:NEWNODE

[END OF IF]

1. SET HEAD:=NEWNODE
2. Calculate time complexity SET TIME\_COMPLEXITY= O(n1/2\*n)/2)
3. Calculate space complexity SET SPACE=O(4(n+1))
4. Visualization of doubly linked list.
5. Display result

[END OF IF]

Step 8: Exit

**4.3.6.2.1.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.1.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.2 Insertion at end**

**4.3.6.2.2.1 Input**

Array of numbers

**4.3.6.2.2.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. CREATE A NEW NODE SET NEWNODE:=new NODE
2. INPUT ITEM
3. SET NEWNODE->DATA:=ITEM
4. SET NEWNODE->LEFT:=NULL
5. SET NEWNODE->RIGHT:=NULL
6. IF(HEAD==NULL)THEN

SET HEAD:=NEWNODE

ELSE

1. SET TEMP:=HEAD
2. REPEAT WHILE(TEMP->RIGHT!=NULL)

SET TEMP”=TEMP->RIGHT

[END OF WHILE]

1. SET TEMP->RIGHT:=NEWNODE
2. SET NEWNODE->LEFT:TEMP

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY= O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4(n+1))
3. Visualization of doubly linked list.
4. Display result

[END OF IF]

Step 7: Exit

**4.3.6.2.2.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.2.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.2.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.3 Insertion at specified position**

**4.3.6.2.3.1 Input**

Array of numbers

**4.3.6.2.3.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. Create a NODE SET newnode:= new NODE;
2. SET newnode->data=element;
3. SET temp:=head;
4. IF((pos=0)&&(head==NULL))THEN
5. SET newnode->link:=NULL;
6. SET head:=newnode;

ELSE IF((pos=0)&&(head!=NULL))

1. newnode->link=head;
2. head:=newnode;

ELSE

1. REPEAT STEP FOR i=0 TO i<pos-1

SET temp:=temp->link;

[END OF FOR]

1. IF(temp==NULL)THEN

WRITE: “Node in the list are less than position”

ELSE

1. SET newnode->link:=temp->link;
2. SET temp->link:=newnode;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY= O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4(n+1))
3. Visualization of doubly linked list.
4. Display result

[END OF IF]

Step 7: Exit

**4.3.6.2.3.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.3.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.3.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.4 Deletion at beginning**

**4.3.6.2.4.1 Input**

Array of numbers

**4.3.6.2.4.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF(HEAD==NULL) THEN

WRITE:” linked list is empty”

ELSE

1. SET ITEM: =HEAD->DATA
2. WRITE: ITEM
3. SET TEMP: =HEAD
4. SET HEAD: =HEAD->RIGHT
5. IF (HEAD! =NULL) THEN

SET HEAD->LEFT: NULL

[END OF IF]

1. DELETE(TEMP)

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY= O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4(n+1))
3. Visualization of doubly linked list.
4. Display result

[END OF IF]

Step 2: Exit

**4.3.6.2.4.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.4.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.4.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.5 Deletion at end**

**4.3.6.2.5.1 Input**

Array of numbers

**4.3.6.2.5.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF(HEAD==NULL) THEN

WRITE:” linked list is empty”

ELSE

1. IF(HEAD->RIGHT! =NULL) THEN
2. SET TEMP1: =HEAD
3. REPEAT WHILE(TEMP1->RIGHT! =NULL)

SET TEMP1: =TEMP1->RIGHT

[END OF WHILE]

1. SET TEMP2: =TEMP1->LEFT
2. SET TEMP2->RIGHT: NULL
3. SET ITEM: =TEMP1->DATA
4. WRITE: ITEM
5. DELETE(TEMP1)

ELSE

1. SET ITEM: =HEAD->DATA
2. WRITE: ITEM
3. DELETE(HEAD)
4. SET HEAD: =NULL

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY= O((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4(n+2))
3. Visualization of doubly linked list.
4. Display result

[END OF IF]

Step 2: Exit

**4.3.6.2.5.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.5.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.5.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.6 Deletion at specified position**

**4.3.6.2.6.1 Input**

Array of numbers

**4.3.6.2.6.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF (head==NULL) THEN

WRITE: “Linked list is empty”

ELSE

1. SET temp1: =NULL
2. SET temp 2: =head;
3. REPEAT STEP i & ii

WHILE (temp2->link! =NULL)

SET temp1: =temp2;

(ii) SET teemp2: =temp2->link;

[END OF WHILE]

1. WRITE: “deleted element is:” temp2->data;
2. Delete(temp2);
3. IF (temp1! =NULL) THEN

SET temp1->link: =NULL;

ELSE

SET head: =NULL;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O ((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4n+8)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 2: Exit

**4.3.6.2.6.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.6.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.6.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.6.2.7 Deletion on element**

**4.3.6.2.7.1 Input**

Array of numbers

**4.3.6.2.7.2 Procedural detail**

**Algorithm:**

Step 1: start

Step 2: Input number

Step 3: IF not validate THEN

goto step 2

ELSE

1. IF(head==NULL)THEN

WRITE: “Linked list is empty”

ELSE

1. SET temp1:=NULL
2. SET temp 2:=head;
3. REPEAT STEP i & ii

WHILE(temp2->link!=NULL)

SET temp1:=temp2;

(ii) SET teemp2:=temp2->link;

[END OF WHILE]

1. WRITE: “deleted element is:”temp2->data;
2. Delete(temp2);
3. IF(temp1!=NULL)THEN
4. SET temp1->link:=NULL;

ELSE

SET head:=NULL;

[END OF IF]

[END OF IF]

1. Calculate time complexity SET TIME\_COMPLEXITY=O ((n1/2\*n)/2)
2. Calculate space complexity SET SPACE=O(4n+8)
3. Visualization of linked list.
4. Display result

[END OF IF]

Step 4: Exit

**4.3.6.2.7.3 File Input/Output Interfaces**

Style.css

**4.3.6.2.7.4 Output**

Displays the Doubly linked list with time and space complexity.

**4.3.6.2.7.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.7 Tree**

**4.3.7.1 Binary search tree**

**4.3.7.1.1 Insertion**

**4.3.7.1.1.1 Input**

Number

**4.3.7.1.1.2 Procedural detail**

Insertion

Input module

Input number

validation

Number>root

Insert at right

Insert at left

Display modified tree

**Structure chart:**

Fig 4.19 Insertion

**4.3.7.1.1.3 File input output interfaces**

Style.css file

**4.3.7.1.1.4 Output**

Displays the binary tree including the new element also calculates the time and space complexity.

**4.3.7.1.1.5 Implementation**

Textbox, label, button.

**4.3.7.1.2 Deletion**

**4.3.7.1.2.1 Input**

Number

**4.3.7.1.2.2 Procedural detail**

**Structure chart:**

Deletion

Input module

Input number

validation

Delete number

Display modified tree

Fig 4.20 Deletion

**4.3.7.1.2.3 File input output interfaces**

Style.css file

**4.3.7.1.2.4 Output**

Displays the binary tree excluding the removed element also calculates the time and space complexity.

**4.3.7.1.2.5 Implementation**

Textbox, label, button.

**4.3.7.1.3 Searching**

**4.3.7.1.3.1 Input**

Number

**4.3.7.1.3.2 Procedural detail**

Searching

Input module

Input number

validation

Searches number

Display result

**Structure chart:**

Fig 4.21 Searching

**4.3.7.1.3.3 File input output interfaces**

Style.css file

**4.3.7.1.3.4 Output**

Displays the binary tree with searched element position also calculates the time and space complexity.

**4.3.7.1.3.5 Implementation**

Textbox, label, button.

**4.3.7.1.4 Pre-order**

**4.3.7.1.4.1 Input**

Number

**4.3.7.1.4.2 Procedural detail**

**Structure chart**

pre-order

Read tree

Traverse root

Display result

Traverse left sub tree

Traverse right sub tree

Fig 4.22 Pre-order

**4.3.7.1.4.3 File input output interfaces**

Style.css file

**4.3.7.1.4.4 Output**

Displays the binary tree with searched element position also calculates the time and space complexity.

**4.3.7.1.4.5 Implementation**

Textbox, label, button.

**4.3.7.1.5 Post-order**

**4.3.7.1.5.1 Input**

Number

**4.3.7.1.5.2 Procedural detail**

**Structure chart**

post-order

Read tree

Traverse root

Display result

Traverse left sub tree

Traverse right sub tree

Fig 4.23 Post-order

**4.3.7.1.4.3 File input output interfaces**

Style.css file

**4.3.7.1.4.4 Output**

Displays the binary tree with searched element position also calculates the time and space complexity.

**4.3.7.1.4.5 Implementation**

Textbox, label, button.

**4.3.7.1.6 In-order**

**4.3.7.1.6.1 Input**

Number

**4.3.7.1.6.2 Procedural detail**

**Structure chart:**

In-order

Read tree

Traverse left sub tree

Display result

Traverse root

Traverse right sub tree

Fig 4.24 In-order

**4.3.7.1.5.3 File input output interfaces**

Style.css file

**4.3.7.1.5.4 Output**

Displays the list of traversed order.

**4.3.7.1.5.5 Implementation**

Textbox, label, button.

**4.3.8 Graph**

**4.3.8.1.1 BFS**

**4.3.8.1.1 Input**

Enter Source and Destination Node.

**4.3.8.1.2 Procedural detail**

**Flowchart:**

for i=1 to n

for j=1 to n

Q[i]=0 visited[i]=0

A[i][j]=x11

YES

NO

Start

Input array elements

Validation

Display error message

n= no.of vertex v=source vertex ,e=0

Visualizing of BFS algorithm

Time complexity

Best=O(n+e)

average=O(n+e)

worst=O(n+e)

Space complexity

Worst=(n\*n\*2)+4

Display time and space complexity

End

If (f<=r)

visited[v][f]=1 bfs(q[f++])

YES

NO

NO

bfs(v)

for j=1 to n

If A[v][j]&&!visited[i]

Q[++s]=i

YES

Fig 4.25 BFS

**4.3.8.1.3 File Input/Output Interfaces**

Style.css

**4.3.8.1.4 Output**

It will display the complete from Source to Destination Node in given tree.

**4.3.8.1.5 Implementation aspect**

Textbox, Buttons, Labels

**4.3.8.2 DFS**

**4.3.8.2.1 Input**

Enter Source and Destination Node.

**4.3.8.2.2 Procedural detail**

**Flowchart:**

for i=1 to n

for j=1 to n

visited[i]=0

A[i][j]=x11

YES

NO

Start

Input array elements

Validation

Display error message

n= no.of vertex v=source vertex ,e=0

Visualizing of DFS algorithm

Time complexity

Best=O(n+e)

average=O(n+e)

worst=O(n+e)

Space complexity

Worst=(n\*n\*2)+4

Display time and space complexity

End

DFS(v)

for i=1 to n

for j=1 to n

If A[i][j]==0

e++

NO

YES

DFS(v)

k=k+1, visited[i]=1

for j=1 to n

If A[i][j] ==1

&& visited[j]==0

DFS[j]

NO

YES

Fig 4.26 DFS

**4.3.8.2.3 File Input/Output Interfaces**

Style.css

**4.3.8.2.4 Output**

It will display the complete from Source to Destination Node in given tree.

**4.3.8.2.5 Implementation aspect**

Textbox, Buttons, Labels