**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.

**2.2.1.4 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.

* Specify the end user minimum characteristics or knowledge require to 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.
   * + 1. **Shell Sort**
7. **Input**: Array of numbers
8. **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.
9. **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 color 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 color 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