**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**

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

3

Insertion at begin

Insertion at end

Insertion at position

Deletion at begin

Deletion at end

Deletion at position

4

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

**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

**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

**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

**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

**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:**

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

Visualizing of counting sort algorithm

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

**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)

**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

**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

**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

**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

**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

**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

* + - * 1. **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

**4.3.3.1.3 File Input/Output Interface**

Style.CSS

* + - * 1. **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

**4.3.3.2.3 File Input/Output Interface**

Style.CSS

* + - * 1. **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

**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:**

**4.3.3.4.3 File Input/Output Interface**

Style.CSS

* + - * 1. **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

**4.3.3.5.3 File Input/Output Interface**

Style.CSS

* + - * 1. **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

**4.3.3.6.3 File Input/Output Interface**

Style.CSS

* + - * 1. **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 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 6: 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 5: 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 2: 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**

**Structure chart:**

Insertion

Input module

Input number

validation

Number>root

Insert at right

Insert at left

Display modified tree

**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

**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**

**Structure chart:**

Searching

Input module

Input number

validation

Searches number

Display result

**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

**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 In-order**

**4.3.7.1.5.1 Input**

Number

**4.3.7.1.5.2 Procedural detail**

**Structure chart:**

In-order

Read tree

Traverse left sub tree

Display result

Traverse root

Traverse right sub tree

**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 bubble sort 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

**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 bubble sort 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

**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