Data Structure and Algorithms

Session-16

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Time Complexity - Declaring, Instantiating, Initializing a 1D Array:

✓ **Instantiation of an Array:**

- ✓ arrayRefVar=new datatype[size]; ---------- O(1)
- ✓ Example: arr = new int[5]

✓ **Initialization:**

- ✓ arr[0]=10; ---------- O(1)
- ✓ arr[1]=20; ---------- O(1)
- ✓ arr[2]=30; ---------- O(1)
- ✓ arr[3]=40; ---------- O(1)
- ✓ arr[4]=50; ---------- O(1)

---------- O(n)

Time Complexity - Inserting a value in 1D Array:

```
Insert(arr, valueToBeInserted, location):

  if (arr[location] is occupied) ------------------------------------------ O(1)
                                                                            +
      return error // location is already occupied ------------------------- O(1)

  else ---------------------------------------------------------------------- O(1)

      arr[location] = valueToBeInserted ---------------------------------- O(1)
```

$$\left.\begin{array}{l} \text{if (arr[location] is occupied)} \\ \quad \text{return error // location is already occupied} \\ \text{else} \\ \quad \text{arr[location] = valueToBeInserted} \end{array}\right\} \text{O(1)}$$

Time Complexity - Traversing a given 1D Array:

```
TraverseArray(arr):

    loop: i = o to arr.length  ---------------------------- O(n)

        print arr[i] ------------------------------------- O(1)
```

Total Time Complexity - O(n)

Space Complexity – O(1)

Time Complexity - Accessing given cell# of 1D Array:

```
AccessingCell(arr, cellNumber):

    if (cellNumber > sizeof(arr))  ------------------------------------------------------------------------ O(1)

        return exception //cell number cannot be bigger than size of Array ------------------ O(1)

    else ------------------------------------------------------------------------------------------------------ O(1)

      return arr[cellNumber] ------------------------------------------------------------------------------ O(1)
```

Total Time Complexity = O(1)

Space Complexity – O(1)

Time Complexity - Searching a given value in 1D Array:

```
SearchInAnArray(arr, valueToSearch):

    loop: i = 0 to arr.length ------------------------------------------------ O(n)

        if (arr[i] equals valueToSearch) ---------------------------------- O(1)

            return i ------------------------------------------------------------ O(1)

    return error// value not found ------------------------------------------ O(1)
```

Total Time Complexity - O(n)

Space Complexity – O(1)

Time Complexity - Deleting a given value from 1D Array:

```
DeletingValueFromArray(arr, location):

    if (arr[location] is occupied) ------------------------------------ O(1)

        arr[location] = Integer.MinValue ------------------------- O(1)

    else ------------------------------------------------------------------ O(1)

        return // location is already blank ------------------------- O(1)
```

Total Time Complexity = O(1)

Space Complexity – O(1)

Time/Space Complexity of 1D Array:

Particulars	Time Complexity	Space Complexity
Creating an empty Array	O(1)	O(n)
Inserting a value in an array	O(1)	O(1)
Traversing a given Array	O(n)	O(1)
Accessing given cell#	O(1)	O(1)
Searching a given value	O(n)	O(1)
Deleting a given value	O(1)	O(1)

10	20	30	40	50		

Declaring, Instantiating, Initializing a 2D Array:

✓ **Instantiation of an Array:**

- ✓ arrayRefVar = new datatype[row][col]; ---------- O(1)
- ✓ Example: arr = new int[2][3]

✓ **Initialization:**

- ✓ a[0][0]=10; ---------- O(1)
- ✓ a[0][1]=20; ---------- O(1)
- ✓ a[0][2]=30; ---------- O(1)
- ✓ a[1][0]=40; ---------- O(1)
- ✓ a[1][1]=50; ---------- O(1)
- ✓ a[1][2]=60; ---------- O(1)

---------- O(mn)

Time Complexity - Inserting a value in 2D Array:

```
Insert(arr, valueToBeInserted, rowNumber, colNumber):

  if (arr[rowNumber][colNumber] is occupied) ---------------------------------- O(1)

      return error // location is already occupied  ----------------------------- O(1)

  else  ------------------------------------------------------------------------------ O(1)

      arr[rowNumber][colNumber] = valueToBeInserted ----------------------- O(1)
```

Time Complexity = O(1)

Space Complexity = O(1)

Time Complexity - Traversing a given 2D Array:

```
TraverseArray(arr):

    loop: row = o to rows   ----------------------- O(m)

        loop: col = o to col ------------------------ O(n)

            print arr[row][col] -------------------- O(n)
```

Time Complexity = O(mn)

Space Complexity = O(1)

Time Complexity - Accessing given cell's value of 2D Array:

```
AccessingCell(arr, rowNumber, colNumber):

    return arr[rowNumber][colNumber] ----------+---------- O(1)
```

Time Complexity = O(1)

Space Complexity = O(1)

Time Complexity - Searching a given value in 2D Array:

```
SearchInAnArray(arr, valueToSearch):

    loop: row = o to rows  ------------------------------------------------ O(m)

        loop: col = o to col ------------------------------------------------ O(n)

            if (arr[row][col] equals valueToSearch)  ---------------- O(1)

                print (row,col);return; ------------------------------------ O(1)

    print (value not found) ---------------------------------------------- O(1)

    return  ------------------------------------------------------------- O(1)
```

Time Complexity = O(mn)

Space Complexity = O(1)

Time Complexity - Deleting a given cell's value from 2D Array:

```
DeletingValueFromArray(arr, rowNumber, colNumber):

    arr[rowNumber][colNumber]  = Integer.MIin_Value     ------------------------ O(1)
```

Time Complexity = O(1)

Space Complexity = O(1)

Time/Space Complexity of 2D Array:

Particulars	Time Complexity	Space Complexity
Creating an Array	O(1)	O(mn)
Inserting a value	O(1)	O(1)
Traversing given Array	O(mn)	O(1)
Accessing given cell#	O(1)	O(1)
Searching a given value	O(mn)	O(1)
Deleting a given cell's value	O(1)	O(1)

Time Complexity - Creation of Single Linked List:

```
CreateSingleLinkedList(nodeValue):

    create a head, tail pointer ---------------------------- O(1)

    create a blank node ---------------------------------- O(1)

    node.value = nodeValue ----------------------------- O(1)

    node.next = null ------------------------------------- O(1)

    head = node ------------------------------------------ O(1)

     tail = node ----------------------------------------- O(1)
```

Time Complexity – O(1)

Space Complexity – O(1)

Time Complexity - Insertion in Single Linked List:

```
InsertInLinkedList(head, nodeValue, location):

    create a blank node ---------------------------------------- O(1)

    node.value = nodeValue ------------------------------------- O(1)

  if (!existsLinkedList(head)) ------------------------------- O(1)

      return error //Linked List does not exists ------------- O(1)

  else if (location equals 0) //insert at first position ------ O(1)

      node.next = head ------------------------------------------ O(1)

      head = node ----------------------------------------------- O(1)

 else if (location equals last) //insert at last position ----- O(1)

      node.next = null ------------------------------------------ O(1)

      last.next = node ------------------------------------------ O(1)

      last = node   //to keep track of last node ---------------- O(1)

  else //insert at specified location ------------------------- O(1)

    loop: tmpNode = 0 to location-1  //loop till we reach specified node and end the loop ---------- O(n)

    node.next = tmpNode.next ----------------------------------- O(1)

    tmpNode.next = node ---------------------------------------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Traversal of Single Linked List:

```
TraverseLinkedList (head):

    if head == NULL, then return ---------------------------------------------------------------------------- O(1) +

    loop: head to tail ------------------------------------------------------------------------- O(n)  ]  ---------------- O(n)
                                                                                                        }
        print currentNode.Value ------------------------------------------- O(1)                        ]
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Searching a node in Single Linked List:

```
SearchNode(head, nodeValue):

   loop: tmpNode = start to tail ------------------------------------------------------------------ O(n)

      if (tmpNode.value equals nodeValue) ---------------------- O(1)

         print tmpNode.Value //node value found -------------- O(1)

          return ------------------------------------------------------ O(1)

    return //nodeValue not found ------------------------------------------------------------------------------------------------------ O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Deletion of node from Single Linked List:

```
DeletionOfNode(head, Location):
    if (!existsLinkedList(head)) ---------------------------------------------------------- O(1)
        return error //Linked List does not exists ------------------------------------------ O(1)
    else if (location equals 0)  //we want to delete first node ------------------------------ O(1)
        head = head.next -------------------------------------------------------------------- O(1)
        if this was the only element in list, then update tail = null ----------------------- O(1)
    else if (location >= last) ---------------------------------------------------------------- O(1)
        if (current node is only node in list) then, head = tail = null; return; ------------ O(1)
        loop till 2nd last node (tmpNode) --------------------------------------------------- O(n)
        tail = tmpNode; tmpNode.next = null ------------------------------------------------- O(1)
    else // if any internal node needs to be deleted ------------------------------------------ O(1)
        loop: tmpNode = start to location-1 ------------------------------------------------- O(n)
        tmpNode.next = tmpNode.next.next  //delete the required node ------------------------ O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Deletion of entire Single Linked List:

DeleteLinkedList(head, tail):

head =null ---------- O(1)

tail = null ---------- O(1)

Time Complexity – O(1)

Space Complexity – O(1)

Time Complexity - Creation of Circular Single Linked List:

CreateSingleLinkedList (nodeValue):

create a blank node ---------- O(1)

node.value = nodeValue ---------- O(1)

node.next = node ---------- O(1)

head = node ---------- O(1)

tail = node ---------- O(1)

Time Complexity – O(1)

Space Complexity – O(1)

Time Complexity - Insertion in Circular Single Linked List:

```
InsertInLinkedList(head, nodeValue, location):
    create a blank node ---------------------------------------- O(1)
    node.value = nodeValue ------------------------------------ O(1)
    if (!existsLinkedList(head)) ------------------------------ O(1)
        return error //Linked List does not exists ------------ O(1)
    else if (location equals 0) //insert at first position ---- O(1)
        node.next = head ------------------------------------- O(1)
        head = node ------------------------------------------ O(1)
        next = head ------------------------------------------ O(1)
    else if (location equals last) //insert at last position -- O(1)
        node.next = head ------------------------------------- O(1)
        tail.next = node ------------------------------------- O(1)
        tail = node   //to keep track of last node ----------- O(1)
    else //insert at specified location ------------------------ O(1)
    loop: tmpNode = 0 to location-1  //loop till we reach specified node and end the loop -------- O(n)
    node.next = tmpNode.next ----------------------------------- O(1)
    tmpNode.next = node ---------------------------------------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Traversal of Circular Single Linked List:

```
TraverseLinkedList (head):

   if head == NULL, then return ------------------------------------------------------------------------------------------ O(1)

   loop: head to tail ------------------------------------------------------------------------------ O(n)  }  ------------------ O(n)

         print currentNode.Value ---------------------------------------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Searching a node in Circular Single Linked List:

```
SearchNode(head, nodeValue):
   loop: tmpNode = start to tail ------------------------------------------------------------------ O(n)
      if (tmpNode.value equals nodeValue) ----------------------- O(1)
         print tmpNode.Value //node value found --------------- O(1)
         return ------------------------------------------------------ O(1)
   return //nodeValue not found ------------------------------------------------------------------------------------------------------ O(1)
```

+ --------- O(1)

---------- O(n)

Time Complexity – O(n)

Space Complexity – O(1)

Time Complexity - Deletion of node from Circular Single Linked List:

```
DeletionOfNode(head, Location):
    if (!existsLinkedList(head)) ---------------------------------------------------------------------------------------------------------- O(1)
        return error //Linked List does not exists ------------------------------------------------------------------------------------------ O(1)
    else if (location equals 0)  //we want to delete first element ------------------------------------------------------------------------ O(1)
        head = head.next; tail.next = head ------------------------------------------------------------------------------------------------ O(1)
        if this was the only element in list, then update tail = null ---------------------------------------------------------------------- O(1)
    else if (location >= last) ------------------------------------------------------------------------------------------------------------ O(1)
        if (current node is only node in list) then, head = tail = null; return ------------------------------------------------------------ O(1)
        loop till 2nd last node (tmpNode) ------------------------------------------------------------------------------------------------- O(n)
        tail = tmpNode; tmpNode.next = head ------------------------------------------------------------------------------------------------ O(1)
    else // if any internal node needs to be deleted ------------------------------------------------------------------------------------- O(1)
        loop: tmpNode = start to location-1 //we need to traverse till we find the previous location ------------------------------------ O(n)
        tmpNode.next = tmpNode.next.next  //delete the required node  ------------------------------------------------------------------ O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)


Thank you