# 

**Lab 07**

**Name: Nimra**

**Registration No. 199281**

**Data structures and Algorithms**

# 

TASK NO1:-

**Singly Linked List – Unsorted**

**ADD ELEMNETS**

1. Add element at the Tail:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and point it to the new node created*.}

* Else {

Traverse the List to find the last node by applying while loop for N times

Append the new node at the end of the list. }

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

As the singly list is unsorted, the while loop is iterated N times to find the last node, so that the last node can be inserted at the tail. The big O analysis of this function is O(N), N indicates the no. of nodes and no. of iterations.

T(N) = O(N)

1. Add element at the head:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and point it to the new node created*.}

* else {

Declare the head pointer to another ‘ptr’ pointer initialized.

*Initialize the head pointer and point it to the new node created*.}.

direct the ‘ptr’ pointer to the next pointer of the new node}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

As the node is inserted at head and no need to check hole list so, BIG O analysis of the funtion is O(1) so

Therefore,

T(n) = O(1).

(3) Add element at the middle:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and point it to the new node created*.}

* Else {

Traverse the List (n/2 iterations) to find the node using a pointer ‘ptr’and insert the new node after it. Run while loop for N/2 times

Add the new node after the node where ‘ptr’ is pointing .

Assign the next node of the node where ‘ptr’ points and after which new node is inserted to the next pointer of the new node. }

* Function ends

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

The while loop for this function iterates N times. So BIG O analysis of the function is O(N) so

Therfore,

T(n) = O(n).

REMOVE ELEMENTS:-

1. AT HEAD:-

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

initialize a new pointer.

direct the head-> next to the new ‘Pointer’.

delete head.

Assign the new pointer to head}

* Function ends

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

Nothing to be iterated. So BIG O analysis of the function is O(1) So,

Therefore

T(n) = O(1).

1. AT TAIL:-

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

Initialize a new pointer and “point” pointer.

assigning the head pointer to the new node.

Transverse the new node till the tail of the list and point the pointer to second last node while loop will run for n times

Then store the previous node of the tail pointer to the “point” pointer.

delete tail.

direct the point to the tail.}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

While loop iterates for N times delete the tail. So BIG O analysis of the function is O(N) where N is no of nodes.

Therefore,

T(N) = O(N).

1. AT MIDDLE:-

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

Initialize a new pointer and “point” pointer.

assigning the head pointer to the new node.

Transverse the new node to the middle of the list and point the pointer to the node present previous from the middle node, while loop runs for n/2 times

Then store the previous node from the middle node to the “point” pointer.

Store the next of the middle node to the next of the point pointer.

delete middle node(new node).}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As while loop is iterated for N/2 TIMES . So BIG O ANALYSIS of the function is O(N)

Therefore,

T(N) = O(N).

It indicates the worst case time complexity of the function.

1. FINDING ELEMENTS:-

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

initialize a new pointer and a variable of type linked list.

Declare and assign the head to the new node.

Transverse the new node to the Node whose value is to be found.

store the data of node to the new variable.}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

If element to be found is placed at last node than time complexity analysis is the worst one and if at the head it is the best and in the middle it is the average. Big o OF the worst case so

T(N) = O(N)

And for best is

T(N) = O(1).

TASK 2

DOUBLY LINKES LIST:-

1. ADD AT THE HEAD:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer and previous of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and the tail pointer and point them to the new node created*.}

* else {

Assign the head pointer to another pointer “ptr”.

Initialize the head pointer to point the new node.

declare the “ptr” pointer to the next pointer of the new node

assign the ptr pointer to the previous of the new node.

sort the linked list by using merge sort.}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

Time complexity of the function is

Therfore

T(n) = o(1) + o(nlogn) = o(nlogn)

1. ADD AT THE TAIL:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer and previous of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and the tail pointer and point them to the new node created*.}

* else {

Assign the tail pointer to another pointer “ptr”.

Initialize the tail pointer to point the new node.

declare the “ptr” pointer’s next to the new node

assign the point pointer to the previous of the new node.

sort the linked list by merge sort.}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

Time complexity of the function is

Therefore

T(n) = o(1) + o(nlogn) = o(nlogn)

1. ADD AT THE MIDDLE:-

* make a node structure and creates a new node.
* Store the data of to the new node value part.
* Initializing the next pointer and previous of the new node to NULL.
* If {

Check whether there are no nodes in the list, and it is empty then  
*Initialize the head pointer and the tail pointer and point them to the new node created*.}

* else {

direct the head to the ptr

transverse the ptr to the position required by the value of that particular node

Direct the ptr->next to another pointer called “ndptr”

direct the new node to the next of the ptr.

direct the ndptr to the next of the new node.

direct the new node to the previous of the ndptr.

direct the ptr to the previous of the new node.}

* Function Ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

Time complexity of the function is

Therefore

T(n) = o(1) + o(nlogn) = o(nlogn)

DELETING ELEMENTS:-

1. AT THE HEAD.

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

Initialize a new pointer.

direct the next of the head to the new pointer.

direct null to the previous of the new pointer.

delete head.

1. direct the new pointer to the head..}

* Function ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

No loops are to iterated. So BIG O Analysis of the function is O(1)

So, Therefore

T(n) = O(1).

1. AT THE TAIL.

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

Initialize a new pointer.

direct the previous of the tail to the new pointer.

assign null to the next of the new pointer.

delete tail.

direct the new pointer to the tail.}

* Function Ends.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:-

No loops are to iterated. So BIG O Analysis of the function is O(1)

So, Therefore

T(n) = O(1).

1. AT THE middle.

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

initialize a new pointer and “point” pointer.

direct the head to the new node.

transverse new node to the middle of the list and point to the one previous than the middle node while loop for n/2 times

store the previous node of the middle node to the “point” pointer.

Store the next of the middle node to the next of the point pointer.

store the point pointer to the previous of the next of the middle node.

delete middle node(new node).}

* Function ends

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

While loop is iterated for N/2 TIMES . So BIG O ANALYSIS of the function is o(n) so So

T(n) = O(n).

4) FINDING ELEMENTS:-

* If {

List is empty and there are no nodes in the list

Return 0}

* else {

initialize a new pointer and a variable of type linked list.

assign the head to the new node.

Transverse the new node to the node whose value is to be found.

store the data of that node to the variable.}

* Function ends

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

If element to be found is present at the last node than time complexity is the worst and at the head it is the best and in the middle it is the average. Big o is the worst case so

T(n) = o(n).

TASK 3:- ARRAY BASED STACK:-

PUSH:-

* Make a push function with void return type and get the value as an argument//we assume array is declared and top is initialized to -1 already.
* if {

Stack is not full

Place the incoming argument value to the incremented top(index) of the array.}

* End function.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

NO of loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

POP:-

* declare function with void argument but with some specific return type.
* initialize a variable say element with -1.
* if {

stack is not empty

store the top most element in the variable element.

Decrement the top.

Return element.}

* End function.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

NO of loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

PEEK:-

* declare function with void argument but some specific return type.
* initialize a variable say element with -1.
* if {

stack is not empty

store the top most element in the variable element.

Return element.}

* End function.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

NO of loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

IsEMPTY:-

* declare the function with bool return type and void argument.
* if {

Top is less than o.

return 1 }

* end function

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

NO of loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

IsFULL:-

* declare the function with bool return type and void argument.
* if {

Top is equal to one less than the size of the array.

return 1 }

* end function

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

TASK 4:- DOUBLY LINKED LIST BASED QUEUE

ENQUEUE:-

* initalize a new node.
* save the value in the new node’s data part.
* Initialize the next AND PREVIOUS pointer of the new node to NULL.
* If {

There are no nodes in the list

*INITIALIZE THE FRONT and the REAR POINTER TO POINT THE NEW NODE.* }

* else {

Assign the tail pointer to another pointer “point”.

INITIALIZE THE tail POINTER TO POINT THE NEW NODE(TAIL IS INCREMENTED).

assign the “point” pointer’s next to the new node

assign the point pointer to the previous of the new node.

* END FUNCTION.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO of loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

DEQUEUE:-

* ► If {

There are no nodes in the list

RETURN 0}

* else {

generate a new pointer and a variable to store the value.

store the value of the front to the variable

assign the next of the FRONT to the new pointer.

Assign null to the previous of the new pointer.

delete FRONT.

Assign the new pointer to the FRONT (front is incremented).

return the value.}

* END FUNCTION.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

IsEMPTY:-

* declare the function with bool return type and void argument.
* if {

FRONT and REAR are NULL.

return 1 }

* end function

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

IsFULL:-

* declare the function with bool return type and void argument.
* if {

Computer runs out the memory.

return 1 }

* end function

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).

FIRST ELEMENT:-

* declare function with void argument but some specific required return type.
* initialize a variable say element with -1.
* if {

stack is not empty

store the top most element in the variable element.

Return element.}

* End function.

ASYMPTOTIC TIME COMPLEXITY ANALYSIS:

As NO loop NEEDS TO BE ITERATED. So BIG O ANALYSIS of the function is o(1)

So T(n) = O(1).