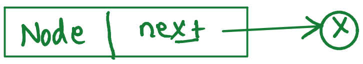
Linked List

07 January 2023 19:22

```
Class Node{
    int value;
    Node next;   //the element next to the current node

    Node (int x)
    {   value = x;
    }
}
```

Node object



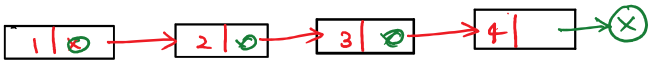
which Node is next to me

Age = value
person standing next = next variable.

Only know the location of head node.

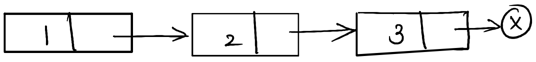
```
class LinkedList{
        Node head;
```

let's say the LinkedList have no element.
Means head node will be null.



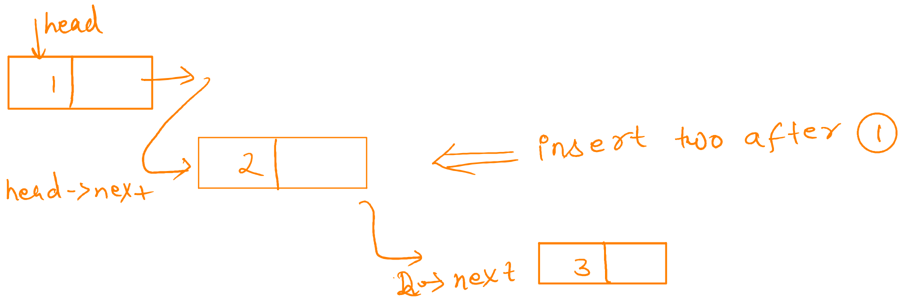
Node has two component.

① Value stored
② next element (who come next)



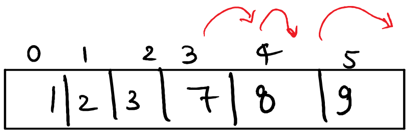
Adding elements in a LL:-

if(head == NULL) → LL has no element;

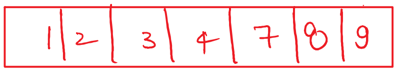


Let's consider an Array :-

Insert 4 at position 3

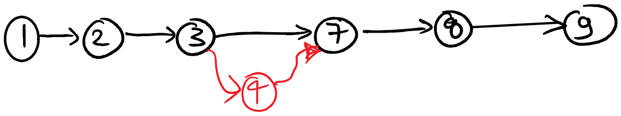


shift my elements -

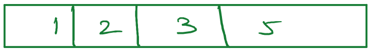


Worst case time complexity will be O(n)

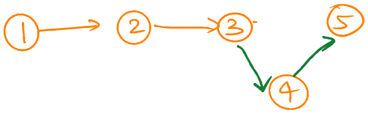
perform same things in Linked List :-



Don't need to shift the element -



store all the element and insert ④ at position ③



Time complexity for this insertion will be

I create an array of size 100 But I am only storing 50 element.

(50) ← Wasted -

LL will Used memory only for the stored element and size increases - dynamicaly as element are added.

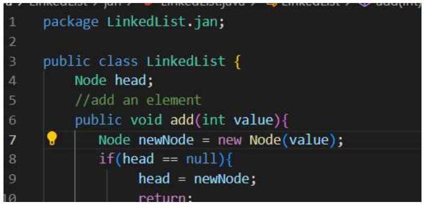
Space for 6 element is occupied -.

Space for only One element will beused.

Linked List class :-

Ist case When List is empty-

```
class LinkedList{
        Node head;
```

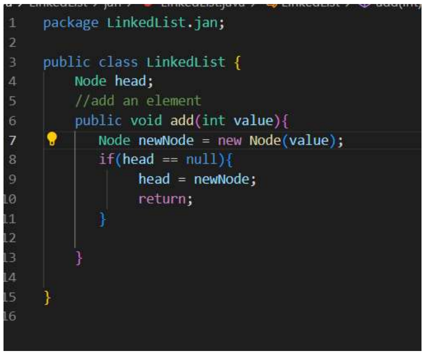


```
package LinkedList.jan;

public class LinkedList {
    Node head;
    //add an element
    public void add(int value){
        Node newNode = new Node(value);
        if(head == null){
            head = newNode;
            return;
```

[illegible] :-

class LinkedList{

Node head;

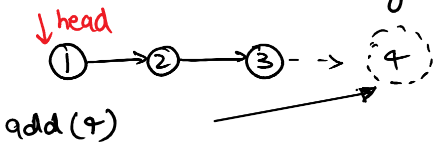


```
package LinkedList.jan;

public class LinkedList {
    Node head;
    //add an element
    public void add(int value){
        Node newNode = new Node(value);
        if(head == null){
            head = newNode;
            return;
        }

    }

}
```

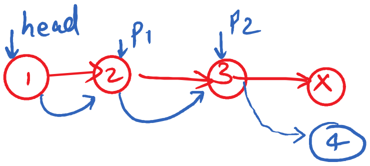
Screen clipping taken: 07-01-2023 20:48

2nd Case :- When List is not empty :-

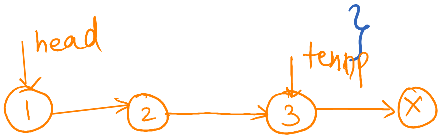
Invert at Tail :-



if I know the location of the head node-
I Can figure out the location of the next node.



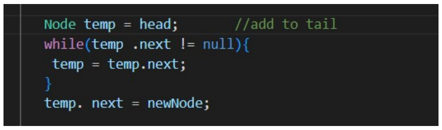
```
Node temp = head
While ( temp. next != null)   // true/false-
    { temp = temp . next ;
    }
```



next · 3 == NULL. → reach the tail-

temp = tail. means temp. next = new Node;

#



```
Node temp = head;        //add to tail
while(temp .next != null){
 temp = temp.next;
}
temp. next = newNode;
```

Screen clipping taken: 07-01-2023 20:58

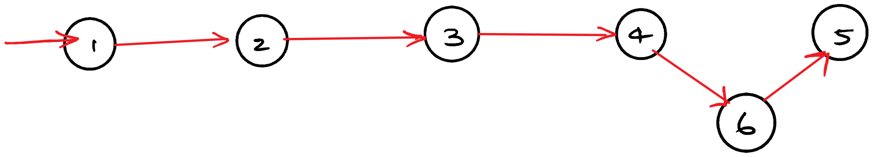
The Time complexity of addition at the tail is O(n)

same as array-

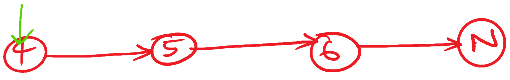
Case:

TC:- for Addition in between for Linked List :-

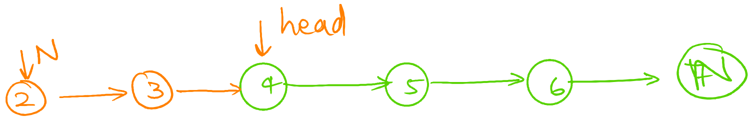
O(n)



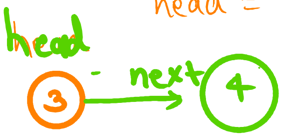
③ Insert at the front :-

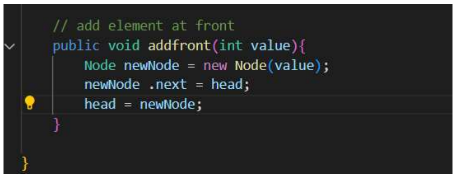


add front(3)   // Add element in the front of the list



Node newNode = new Node(3);
newNode.next = head;
head = newNode;

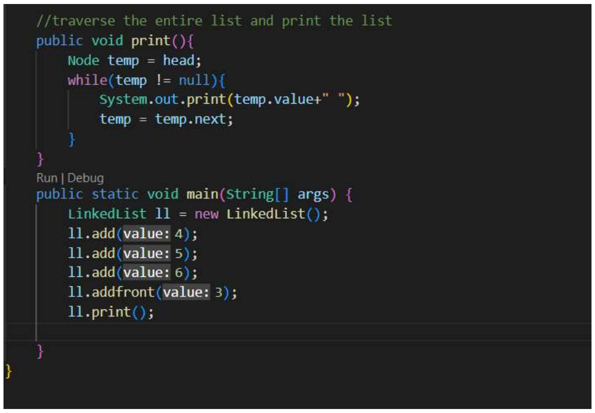




```
// add element at front
public void addfront(int value){
    Node newNode = new Node(value);
    newNode .next = head;
    head = newNode;
}
}
```

So the Time complexity of Add front method is O(1)

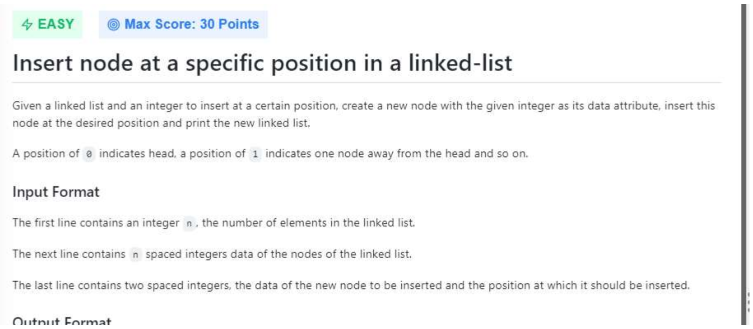
How do I print a Linked List :-



```
    //traverse the entire list and print the list
    public void print(){
        Node temp = head;
        while(temp != null){
            System.out.print(temp.value+" ");
            temp = temp.next;
        }
    }
    Run | Debug
    public static void main(String[] args) {
        LinkedList ll = new LinkedList();
        ll.add(value: 4);
        ll.add(value: 5);
        ll.add(value: 6);
        ll.addfront(value: 3);
        ll.print();

    }
}
```

Screen clipping taken: 07-01-2023 21:21

que :- ①



⚡ EASY   ◎ Max Score: 30 Points

Insert node at a specific position in a linked-list

Given a linked list and an integer to insert at a certain position, create a new node with the given integer as its data attribute, insert this node at the desired position and print the new linked list.

A position of `0` indicates head, a position of `1` indicates one node away from the head and so on.

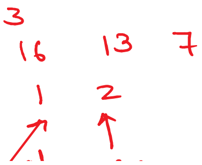
Input Format

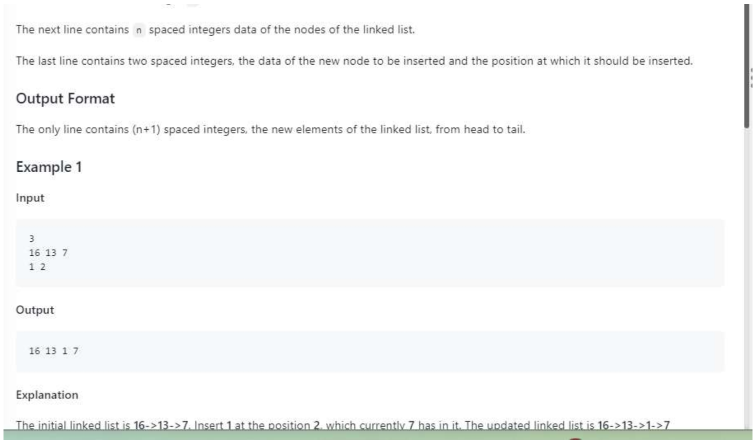
The first line contains an integer `n`, the number of elements in the linked list.

The next line contains `n` spaced integers data of the nodes of the linked list.

The last line contains two spaced integers, the data of the new node to be inserted and the position at which it should be inserted.

Output Format





The next line contains n spaced integers data of the nodes of the linked list.

The last line contains two spaced integers, the data of the new node to be inserted and the position at which it should be inserted.

Output Format

The only line contains (n+1) spaced integers, the new elements of the linked list, from head to tail.

Example 1

Input

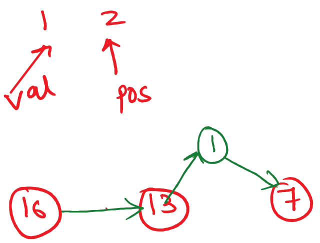
```
3
16 13 7
1 2
```

Output

```
16 13 1 7
```

Explanation

The initial linked list is 16->13->7. Insert 1 at the position 2, which currently 7 has in it. The updated linked list is 16->13->1->7

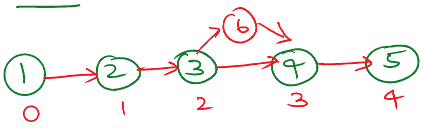


1 is inserted at position 2

Screen clipping taken: 07-01-2023 21:21

① is pos==0 add at front:-

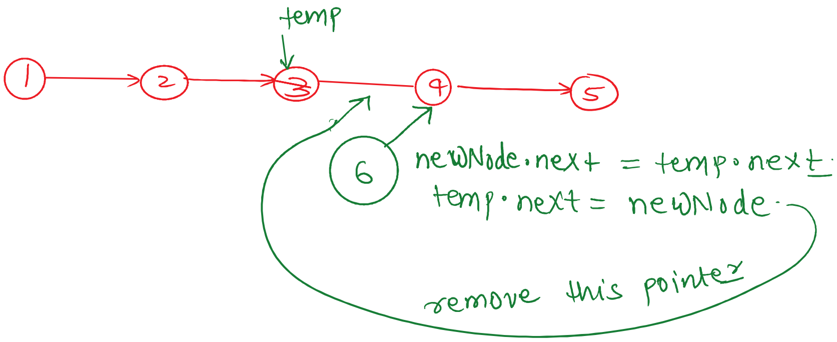
② if pos == ③ for



traverse the

pos=n:-

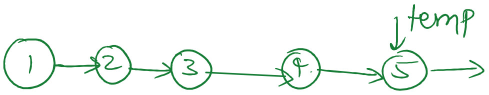
traverse to the node ~~at~~ pos = n-1



newNode.next = temp.next

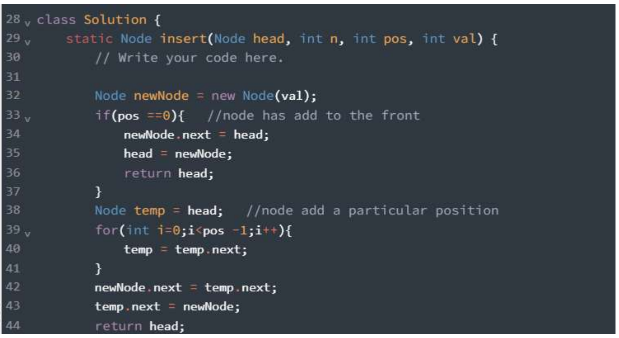
temp.next = newNode

③ node add at last:-

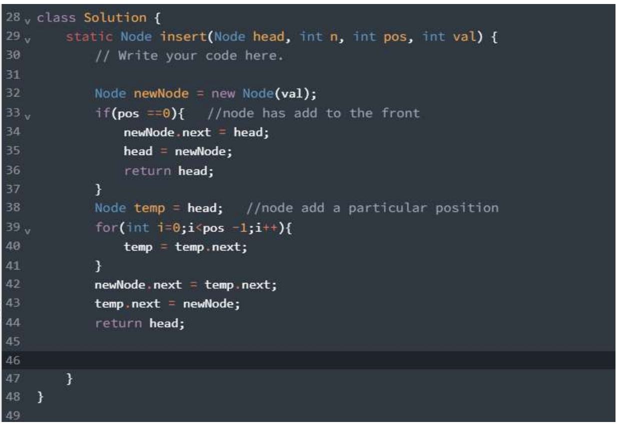


temp.next = new

then last case also it handle in the above



```
class Solution {
    static Node insert(Node head, int n, int pos, int val) {
        // Write your code here.

        Node newNode = new Node(val);
        if(pos ==0){    //node has add to the front
            newNode.next = head;
            head = newNode;
            return head;
        }
        Node temp = head;   //node add a particular position
        for(int i=0;i<pos -1;i++){
            temp = temp.next;
        }
        newNode.next = temp.next;
        temp.next = newNode;
        return head;
```

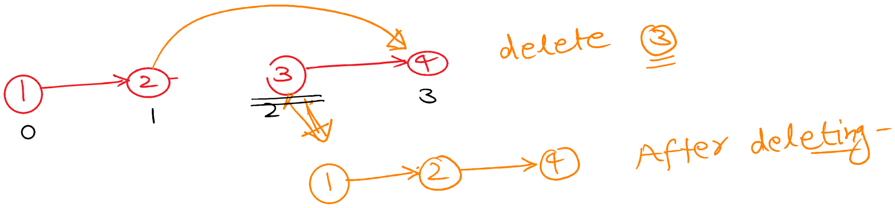


```
class Solution {
    static Node insert(Node head, int n, int pos, int val) {
        // Write your code here.

        Node newNode = new Node(val);
        if(pos ==0){   //node has add to the front
            newNode.next = head;
            head = newNode;
            return head;
        }
        Node temp = head;   //node add a particular position
        for(int i=0;i<pos -1;i++){
            temp = temp.next;
        }
        newNode.next = temp.next;
        temp.next = newNode;
        return head;


    }
}
```

Screen clipping taken: 07-01-2023 21:37

Adding at Any specific position done

Delete nodes :-



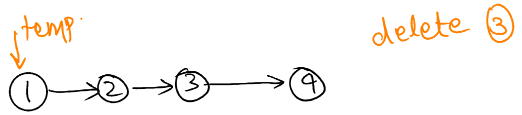
delete node at position x then node at pos x-1 should point at x+1

Two kind of delete method ·

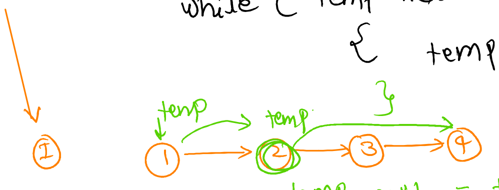
① delete by value - (

② delete by position

#



```
While ( temp. next. Val != val)
        { temp = temp. next
        }
```



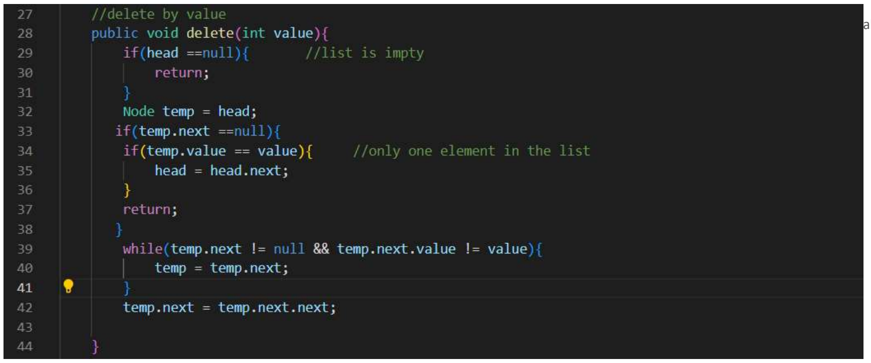
temp.next = temp.next. next.

Edge Case :- Ⓘ if Linked List empty → return

Ⓘⓘ if the size of the Linked List =1

```
if (temp.next == null)
  { if( temp. value == value)
     { head =null;
```

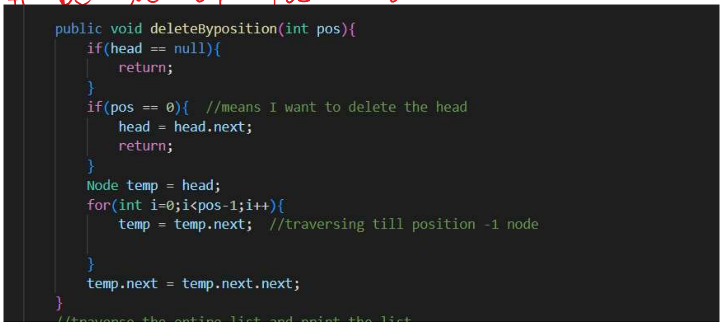
}}



```
    //delete by value
    public void delete(int value){
        if(head ==null){        //list is impty
            return;
        }
        Node temp = head;
       if(temp.next ==null){
        if(temp.value == value){      //only one element in the list
            head = head.next;
        }
        return;
       }
        while(temp.next != null && temp.next.value != value){
            temp = temp.next;
        }
        temp.next = temp.next.next;

    }
```

aken: 07-01-2023 22:12

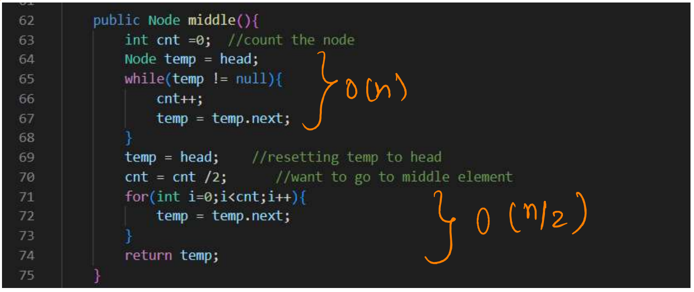
Delete by Position :-



```
    public void deleteByposition(int pos){
        if(head == null){
            return;
        }
        if(pos == 0){  //means I want to delete the head
            head = head.next;
            return;
        }
        Node temp = head;
        for(int i=0;i<pos-1;i++){
            temp = temp.next;  //traversing till position -1 node

        }
        temp.next = temp.next.next;
    }
    //traverse the entire list and print the list
```

Screen clipping taken: 07-01-2023 22:21

Can I find the Middle of LL ?



```
    public Node middle(){
        int cnt =0;  //count the node
        Node temp = head;
        while(temp != null){
            cnt++;
            temp = temp.next;
        }
        temp = head;     //resetting temp to head
        cnt = cnt /2;       //want to go to middle element
        for(int i=0;i<cnt;i++){
            temp = temp.next;
        }
        return temp;
    }
```

} $O(n)$

} $O(n/2)$

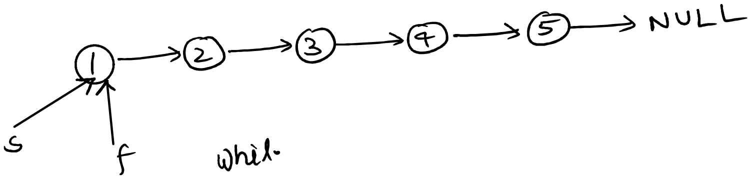
$O(n) + O(\frac{n}{2})$

$\Rightarrow O(\frac{3n}{2})$

Screen clipping taken: 07-01-2023 22:47

What is better solution :- Using slow and fast pointer :-

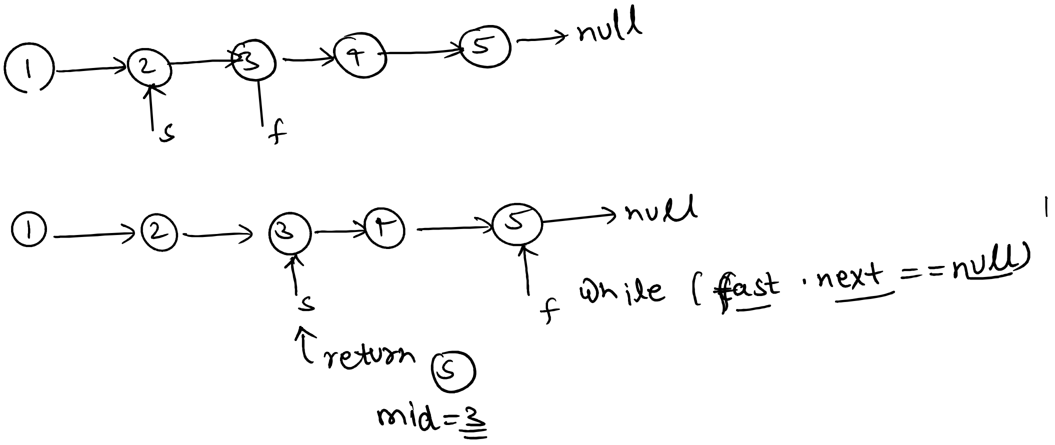
let's say I have the Linked List Like

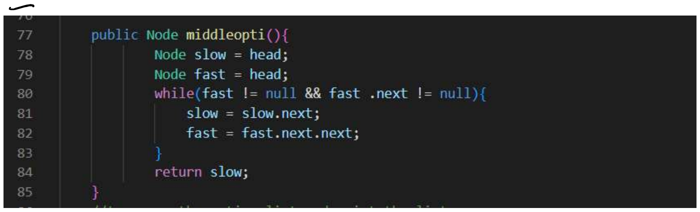


whil-

fast = fast -> next.next ;

slow = slow -> next;

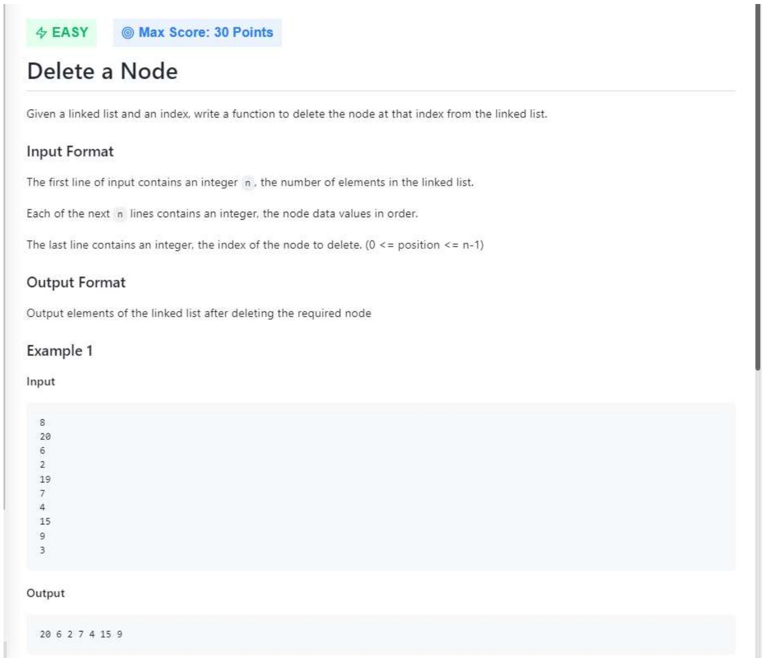




```java
    public Node middleopti(){
            Node slow = head;
            Node fast = head;
            while(fast != null && fast .next != null){
                slow = slow.next;
                fast = fast.next.next;
            }
            return slow;
    }
```

Screen clipping taken: 07-01-2023 22:55

que 2 :-



EASY   Max Score: 30 Points

Delete a Node

Given a linked list and an index, write a function to delete the node at that index from the linked list.

Input Format

The first line of input contains an integer `n`, the number of elements in the linked list.

Each of the next `n` lines contains an integer, the node data values in order.

The last line contains an integer, the index of the node to delete. (0 <= position <= n-1)

Output Format

Output elements of the linked list after deleting the required node

Example 1

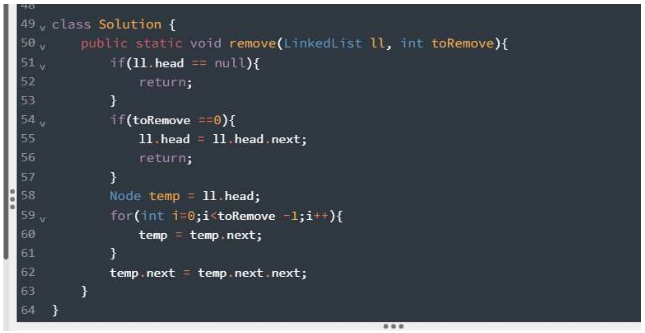
Input

```
8
20
6
2
19
7
4
15
9
3
```

Output

```
20 6 2 7 4 15 9
```

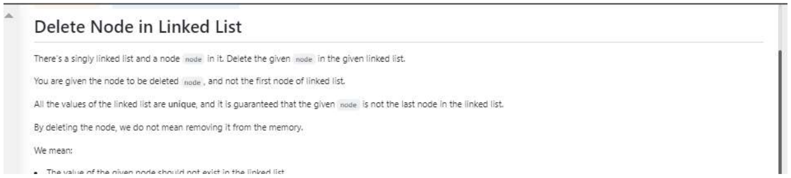
Screen clipping taken: 07-01-2023 23:08



```java
class Solution {
    public static void remove(LinkedList ll, int toRemove){
        if(ll.head == null){
            return;
        }
        if(toRemove ==0){
            ll.head = ll.head.next;
            return;
        }
        Node temp = ll.head;
        for(int i=0;i<toRemove -1;i++){
            temp = temp.next;
        }
        temp.next = temp.next.next;
    }
}
```

Screen clipping taken: 07-01-2023 23:08

que 3 :-



Delete Node in Linked List

There's a singly linked list and a node `node` in it. Delete the given `node` in the given linked list.

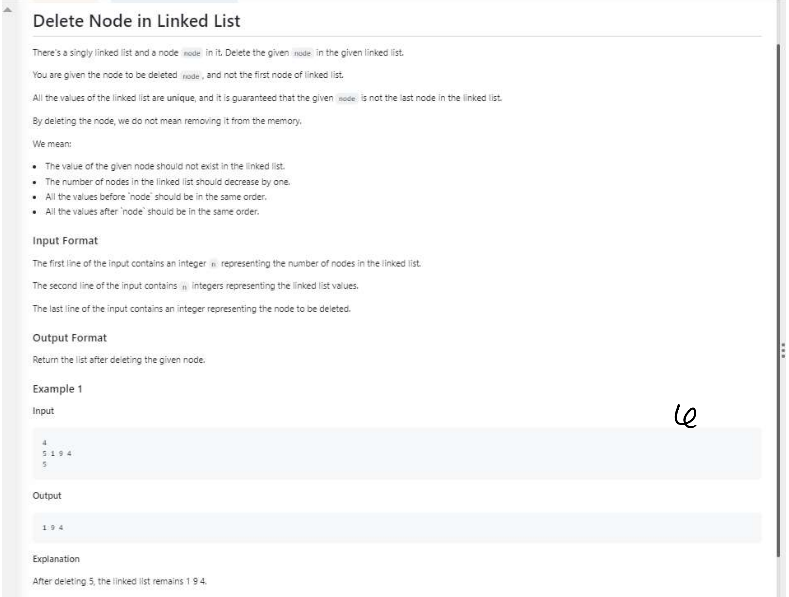
You are given the node to be deleted `node`, and not the first node of linked list.

All the values of the linked list are **unique**, and it is guaranteed that the given `node` is not the last node in the linked list.

By deleting the node, we do not mean removing it from the memory.

We mean:

- The value of the given node should not exist in the linked list



Delete Node in Linked List

There's a singly linked list and a node `node` in it. Delete the given `node` in the given linked list.

You are given the node to be deleted `node`, and not the first node of linked list.

All the values of the linked list are **unique**, and it is guaranteed that the given `node` is not the last node in the linked list.

By deleting the node, we do not mean removing it from the memory.

We mean:

- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before `node` should be in the same order.
- All the values after `node` should be in the same order.

Input Format

The first line of the input contains an integer `n` representing the number of nodes in the linked list.

The second line of the input contains `n` integers representing the linked list values.

The last line of the input contains an integer representing the node to be deleted.

Output Format

Return the list after deleting the given node.

Example 1

Input

```
4
5 1 9 4
5
```

Output

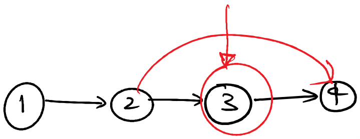
```
1 9 4
```

Explanation

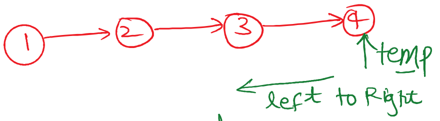
After deleting 5, the linked list remains 1 9 4.

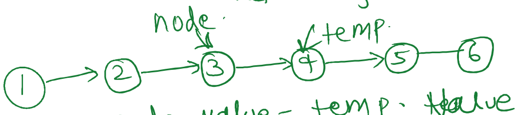
le

Screen clipping taken: 07-01-2023 23:09



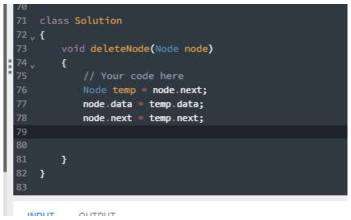
how will I delete it





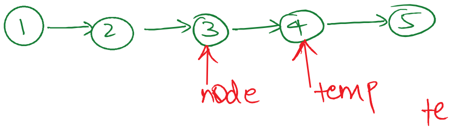
node.value= temp. ~~t~~value

node. next = temp. next



```
class Solution
{
    void deleteNode(Node node)
    {
        // Your code here
        Node temp = node.next;
        node.data = temp.data;
        node.next = temp.next;


    }
}
```

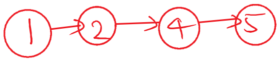
Screen clipping taken: 07-01-2023 23:15



temp = node.next

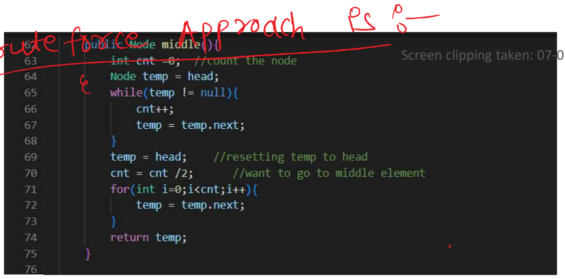
node.value = temp.value

node. next = temp.next



que 4:-

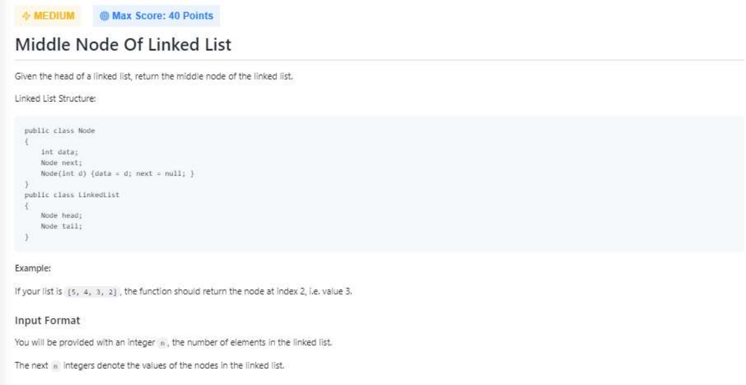
Brute force Approach is:-



```
public Node middle(){
    int cnt =0;  //count the node
    Node temp = head;
    while(temp != null){
        cnt++;
        temp = temp.next;
    }
    temp = head;    //resetting temp to head
    cnt = cnt /2;      //want to go to middle element
    for(int i=0;i<cnt;i++){
        temp = temp.next;
    }
    return temp;
}
```

Screen clipping taken: 07-01-2023 23:20

Screen clipping taken: 07-01-2023 23:19



MEDIUM Max Score: 40 Points

Middle Node Of Linked List

Given the head of a linked list, return the middle node of the linked list.

Linked List Structure:

```
public class Node
{
    int data;
    Node next;
    Node(int d) {data = d; next = null; }
}
public class LinkedList
{
    Node head;
    Node tail;
}
```

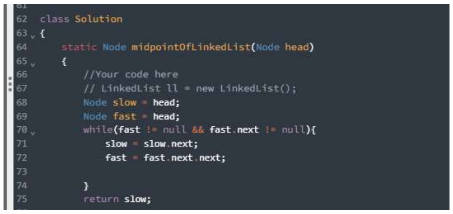
Example:

If your list is `[5, 4, 3, 2]`, the function should return the node at index 2, i.e. value 3.

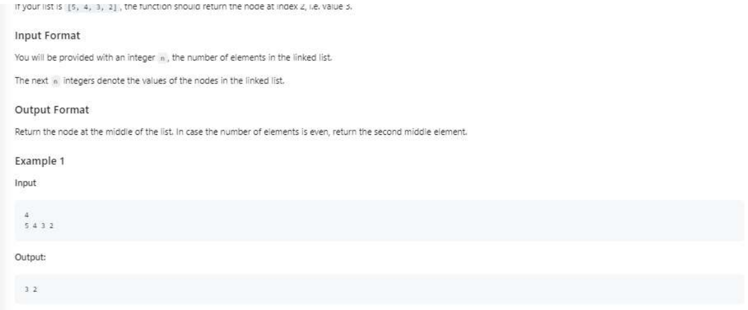
Input Format

You will be provided with an integer `n`, the number of elements in the linked list.

The next `n` integers denote the values of the nodes in the linked list.



```
class Solution
{
    static Node midpointOfLinkedList(Node head)
    {
        //Your code here
        // LinkedList ll = new LinkedList();
        Node slow = head;
        Node fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;

        }
        return slow;
```



If your list is `[5, 4, 3, 2]`, the function should return the node at index 2, i.e. value 3.

Input Format

You will be provided with an integer `n`, the number of elements in the linked list.

The next `n` integers denote the values of the nodes in the linked list.

Output Format

Return the node at the middle of the list. In case the number of elements is even, return the second middle element.

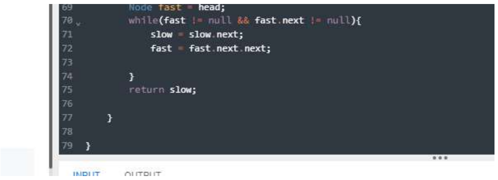
Example 1

Input

```
4
5 4 3 2
```

Output:

```
3 2
```



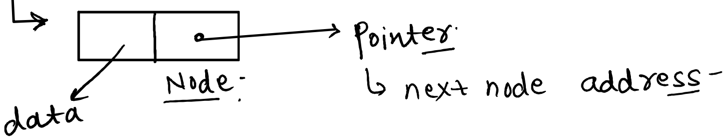
```
        Node fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;

        }
        return slow;

    }

}
```

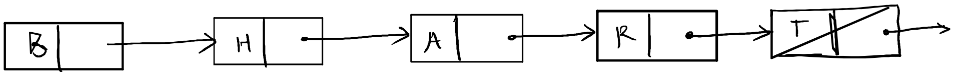
Screen clipping taken: 07-01-2023 23:18

Date - 8-01-2023 :-

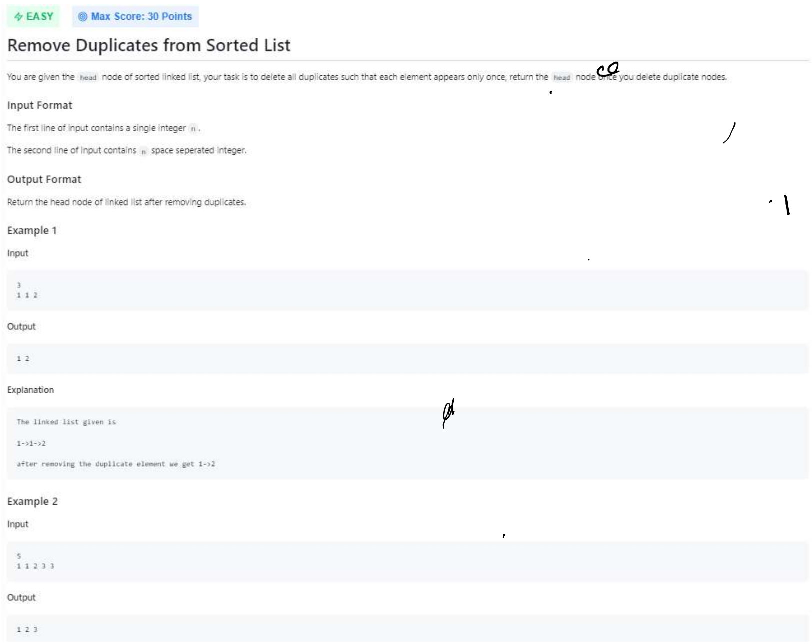
Linked List -II :-

Linear Data structure





que 1 :- Remove Duplicate from Sorted List



Remove Duplicates from Sorted List

You are given the `head` node of sorted linked list, your task is to delete all duplicates such that each element appears only once, return the `head` node once you delete duplicate nodes.

Input Format

The first line of input contains a single integer `n`.

The second line of input contains `n` space seperated integer.

Output Format

Return the head node of linked list after removing duplicates.

Example 1

Input

```
3
1 1 2
```

Output

```
1 2
```

Explanation

```
The linked list given is
1->1->2
after removing the duplicate element we get 1->2
```

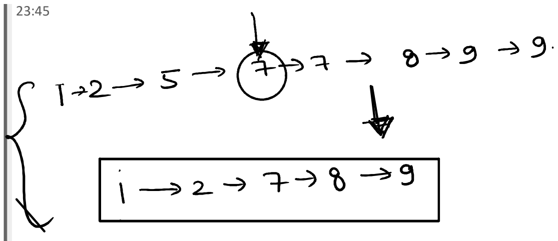
Example 2

Input

```
5
1 1 2 3 3
```

Output

```
1 2 3
```


23:45
1→2→5→7→7→8→9→9.
1→2→7→8→9