

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

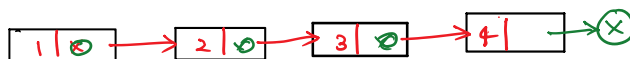
Only know the location of head node -

```
class LinkedList{
```

```
    Node head;
```

let's say the linkedList have no element -

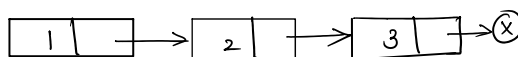
Means head node will be NULL.



Node has two component -

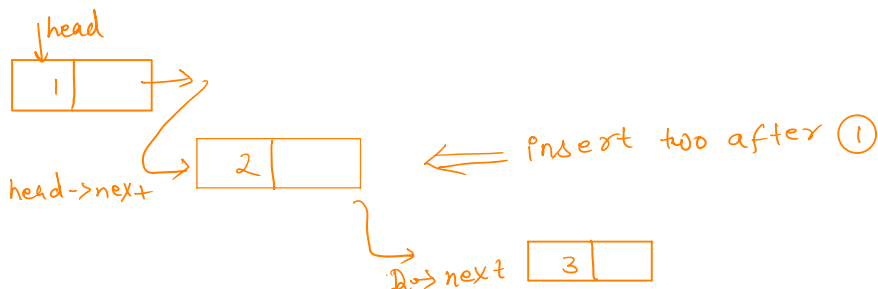
① Value stored

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



Insert 4 at position 3

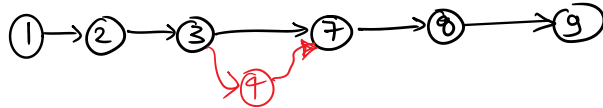
0	1	2	3	4	5
1	2	3	7	8	9

shift my elements -

1	2	3	4	7	8	9
---	---	---	---	---	---	---

Worst case time complexity will be $O(n)$

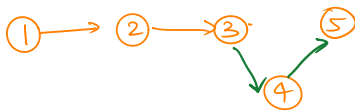
perform same things in Linked List :-



Don't need to shift the element -

1	2	3	5
---	---	---	---

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 use memory only for the stored element and size increases dynamically as element are added -

Space for 6 element is occupied -
Space for only one element will be used -

LinkedList class :-

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

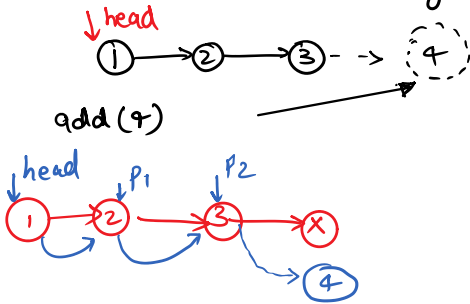
1st case When List is empty -

```
package LinkedList.java;
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 :-
Insert at Tail :-

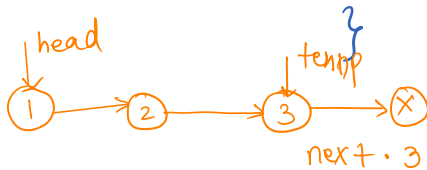
When list is not empty :-



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 of 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;
```

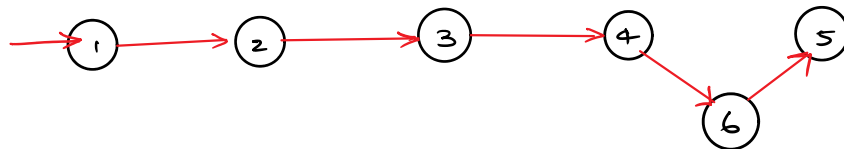
Screen clipping taken: 07-01-2023 20:58

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

same as array-

Case :-

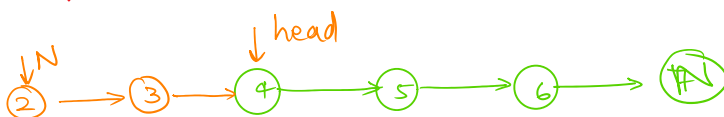
TC :- for Addition in between for Linked List :-
 $O(n)$



③ Insert at the front :-



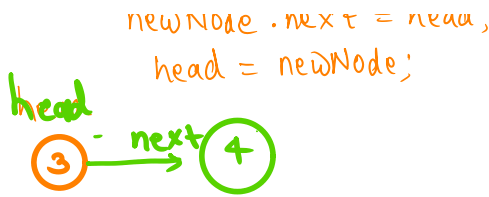
add front(3) // Add element in the front of the list



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



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



```

// 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 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

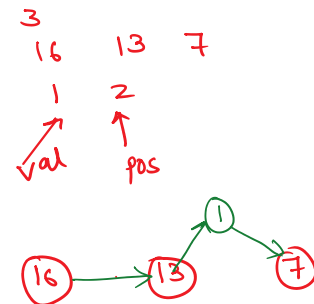
```

16 13 1 7

```

Explanation

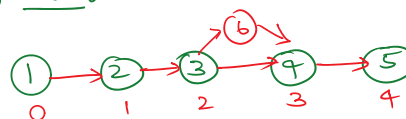
The initial linked list is 16->13->7. Insert 1 at the position 2, which currently 7 has in it. The updated linked list is 16->13->1->7



1 is inserted at position 2

① is pos == 0 add at front:-

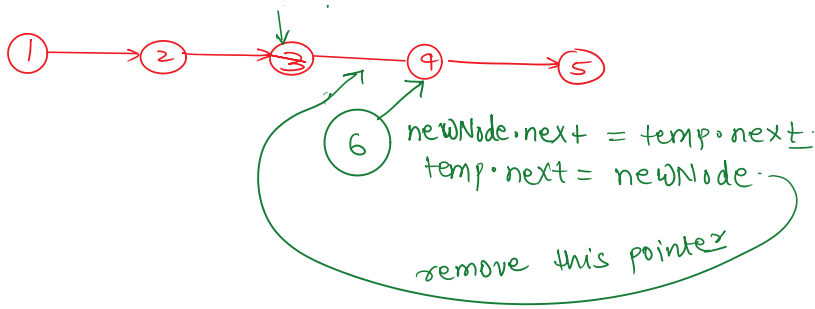
② if pos == 3 for



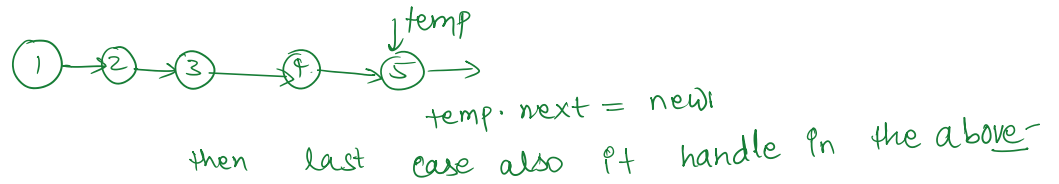
traverse the

pos = n:-

traverse to the node at pos = n-1
temp



⑤ node add at last:-



```

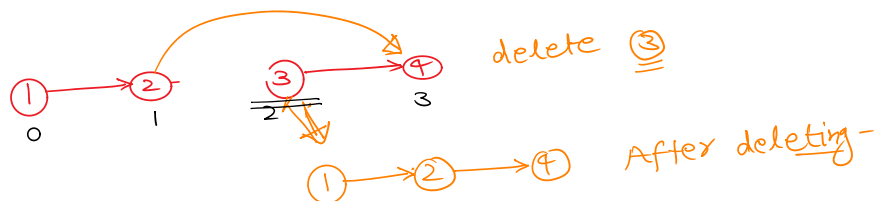
28 class Solution {
29     static Node insert(Node head, int n, int pos, int val) {
30         // Write your code here.
31
32         Node newNode = new Node(val);
33         if(pos == 0){ //node has add to the front
34             newNode.next = head;
35             head = newNode;
36             return head;
37         }
38         Node temp = head; //node add a particular position
39         for(int i=0; i<pos-1; i++){
40             temp = temp.next;
41         }
42         newNode.next = temp.next;
43         temp.next = newNode;
44         return head;
45     }
46 }
47 }
48 }
49

```

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

Adding at Any specific position done

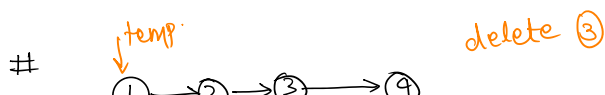
Delete nodes

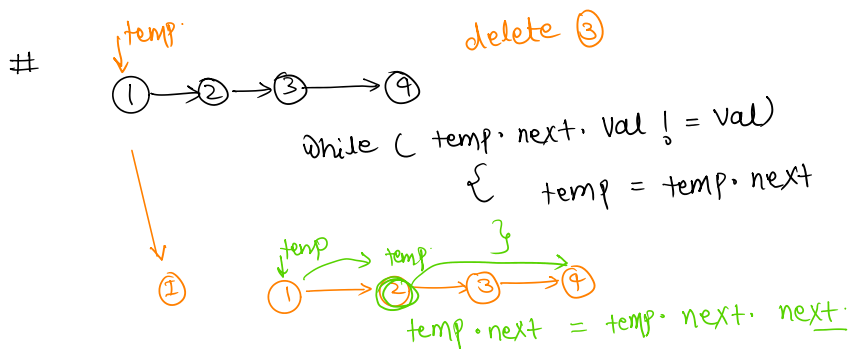


delete node at position x then
node at pos $x-1$ should point at $x+1$

Two kind of delete method -

- ① delete by value - (
- ② delete by position





Edge Case :-

① if Linked List empty → return

② if the size of the Linked List = 1
 if (temp.next == null)
 { if (temp.value == value)
 { head = null;
 }
 }
 }

```

27 //delete by value
28 public void delete(int value){
29     if(head == null){ //list is empty
30         return;
31     }
32     Node temp = head;
33     if(temp.next == null){
34         if(temp.value == value){ //only one element in the list
35             head = head.next;
36         }
37         return;
38     }
39     while(temp.next != null && temp.next.value != value){
40         temp = temp.next;
41     }
42     temp.next = temp.next.next;
43 }
44 
```

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;
}

```

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

Can I find the Middle of LL?

```

62 public Node middle(){
63     int cnt = 0; //count the node
64     Node temp = head;
65     while(temp != null){
66         cnt++;
67         temp = temp.next;
68     }
69     temp = head; //resetting temp to head
70     cnt = cnt / 2; //want to go to middle element
71     for(int i=0; i<cnt; i++){
72         temp = temp.next;
73     }
74     return temp;
75 }

```

$O(n)$

$O(n/2)$

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

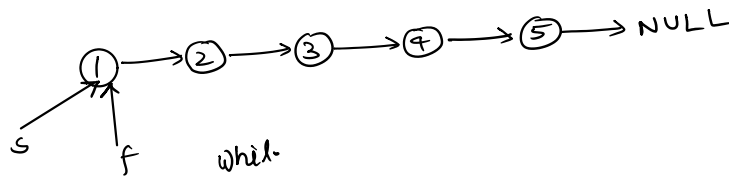
$$\Rightarrow O\left(\frac{3n}{2}\right)$$

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

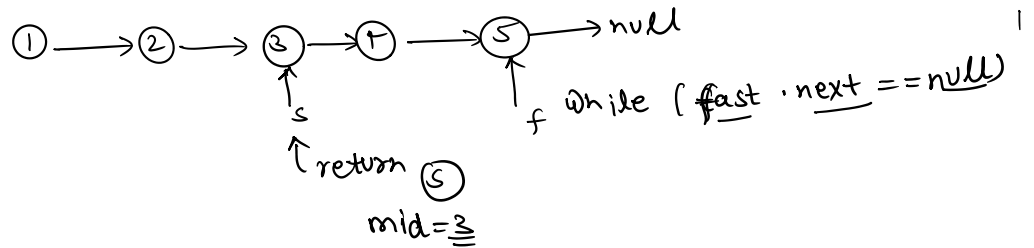
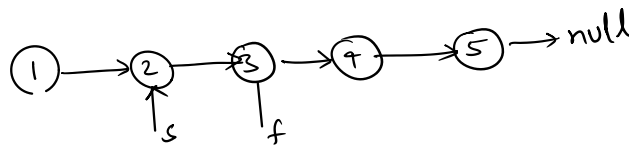
what is better solution :- Using slow and fast pointer :-

what is better solution :- Using slow and fast pointers

let's say I have the Linked List Like



fast = fast -> next.next;
slow = slow -> next;



```

77 public Node middleopti(){
78     Node slow = head;
79     Node fast = head;
80     while(fast != null && fast.next != null){
81         slow = slow.next;
82         fast = fast.next.next;
83     }
84     return slow;
85 }

```

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. ($0 \leq \text{position} \leq n-1$)

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

```

49 class Solution {
50     public static void remove(LinkedList ll, int toRemove){
51         if(ll.head == null){
52             return;
53         }
54         if(toRemove == 0){
55             ll.head = ll.head.next;
56             return;
57         }
58         Node temp = ll.head;
59         for(int i=0; i<toRemove-1; i++){
60             temp = temp.next;
61         }
62         temp.next = temp.next.next;
63     }
64 }

```

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

```

4
15
9
3

Output

20 6 2 7 4 15 9

```

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.
- 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.

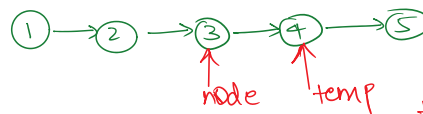
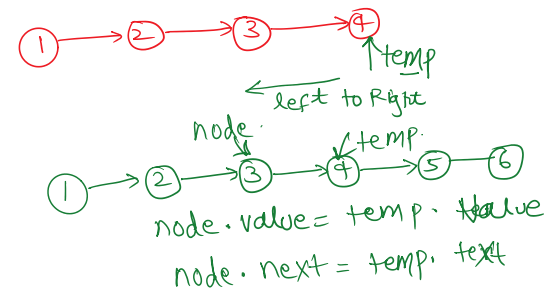
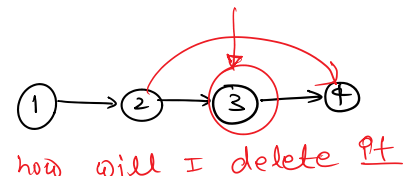
Screen clipping taken: 07-01-2023 23:09

```

76 class Solution
77 {
78     void deleteNode(Node node)
79     {
80         // Your code here
81         Node temp = node.next;
82         node.data = temp.data;
83         node.next = temp.next;
84     }
85 }

```

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



temp = node.next
node.value = temp.value
node.next = temp.next



que 4:-

Brute force Approach Ps 5:-

```

63 public Node middle() {
64     int cnt = 0; //count the node
65     Node temp = head;
66     while(temp != null){
67         cnt++;
68         temp = temp.next;
69     }
70     temp = head; //resetting temp to head
71     cnt = cnt / 2; //want to go to middle element
72     for(int i=0; i<cnt; i++){
73         temp = temp.next;
74     }
75     return temp;
76 }

```

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

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:

```

62 class Solution
63 {
64     static Node midpointOfLinkedList(Node head)
65     {
66         //Your code here
67         // linked list ll = new LinkedList();
68     }
69 }

```



```

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.

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

```

62 class Solution
63 {
64     static Node midpointOfLinkedList(Node head)
65     {
66         //Your code here
67         // linkedlist ll = new LinkedList();
68         Node slow = head;
69         Node fast = head;
70         while(fast != null && fast.next != null){
71             slow = slow.next;
72             fast = fast.next.next;
73         }
74         return slow;
75     }
76 }
77
78
79

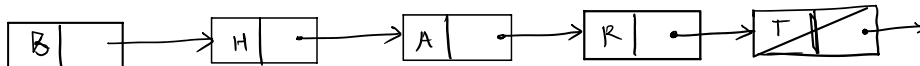
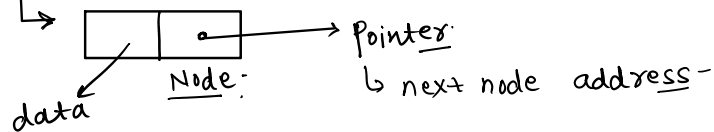
```

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

Date - ~~7~~ - 01-2023 - 8

Linked List - II :-

Linear Data Structure



que 1:- Remove Duplicate from Sorted List

EASY Max Score: 30 Points

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 after you delete duplicate nodes.

Input Format

The first line of input contains a single integer n .

The second line of input contains n space separated integers.

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

