# Delete node Recursively

Send Feedback

Given a singly linked list of integers and position 'i', delete the node present at the 'i-th' position in the linked list recursively.

**Note :**

Assume that the Indexing for the linked list always starts from 0.

No need to print the list, it has already been taken care. Only return the new head to the list.

**input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'i'.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.
0 <= i < M

Time Limit:  1sec

**Sample Input 1 :**

```
1
3 4 5 2 6 1 9 -1
3
```

**Sample Output 1 :**

```
3 4 5 6 1 9
```

```
public class Solution {




	public static LinkedListNode<Integer> deleteNodeRec(LinkedListNode<Integer> head, int pos) {

	//Your code goes here

    if(head==null){

       return head;

    }

    if(pos==0){

       head = head.next;

       return head;
```

```
        }

        else{

            LinkedListNode<Integer>smallerHead = deleteNodeRec(head.next,pos-1);

            head.next = smallerHead;

            return head;

        }

            }



}
```

## Reverse LL (Recursive)

Send Feedback

Given a singly linked list of integers, reverse it using recursion and return the head to the modified list. You have to do this in O(N) time complexity where N is the size of the linked list.

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

*Constraints :*

1 <= t <= 10^2
0 <= M <= 10^4
Where M is the size of the singly linked list.

Time Limit: 1sec

**Sample Input 1 :**

```
1
1 2 3 4 5 6 7 8 -1
```

**Sample Output 1 :**

```
8 7 6 5 4 3 2 1
```

```
public class Solution {



    public static LinkedListNode<Integer> reverseLinkedListRec(LinkedListNode<Integer> head) {

            //Your code goes here
```

```
        if(head==null || head.next==null){
           return head;
        }
        LinkedListNode<Integer>smallerHead = reverseLinkedListRec(head.next);
        LinkedListNode<Integer> node=smallerHead;
        while(node.next !=null){
           node=node.next;
        }
        node.next=head;
        head.next=null;
        return smallerHead;
             }

}
```

## Mid Point Linked List

Send Feedback

For a given singly linked list of integers, find and return the node present at the middle of the list.

*Note :*

```
If the length of the singly linked list is even, then return the first middle node.

Example: Consider, 10 -> 20 -> 30 -> 40 is the given list, then the nodes present at the middle with respective data values are, 20 and 30. We return the first node with data 20.
```

**Input format :**

```
The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.
```

**Remember/Consider :**

```
While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element
```

**Output Format :**

```
For each test case/query, print the data value of the node at the middle of the given list.

Output for every test case will be printed in a seperate line.
```

**Constraints :**

```
1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec
```

**Sample Input 1 :**

```
1
```

```
1 2 3 4 5 -1
```

**Sample Output 1 :**

```
3
```

```
public class Solution {



    public static LinkedListNode<Integer> midPoint(LinkedListNode<Integer> head) {

        //Your code goes here

        if (head==null || head.next==null)

        {

            return head;

        }

        LinkedListNode<Integer> fast=head,slow=head;

        while(fast.next!=null && fast.next.next!=null)

        {

            slow=slow.next;

            fast=fast.next.next;

        }

        return slow;



    }



}
```

## Merge Two Sorted LL

Send Feedback

You have been given two sorted(in ascending order) singly linked lists of integers.

Write a function to merge them in such a way that the resulting singly linked list is also sorted(in ascending order) and return the new head to the list.

**Note :**

```
Try solving this in O(1) auxiliary space.

No need to print the list, it has already been taken care.
```

**Input format :**

```
The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the first sorted singly linked list separated by a single space.
```

The second line of the input contains the elements of the second sorted singly linked list separated by a single space.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output :**

For each test case/query, print the resulting sorted singly linked list, separated by a single space.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t = 10^2
0 <= N <= 10 ^ 4
0 <= M <= 10 ^ 4
Where N and M denote the sizes of the singly linked lists.

Time Limit: 1sec

**Sample Input 1 :**

```
1
2 5 8 12 -1
3 6 9 -1
```

**Sample Output 1 :**

```
2 3 5 6 8 9 12
```

```
ublic class Solution {



    public static LinkedListNode<Integer> mergeTwoSortedLinkedLists(LinkedListNode<Integer> head1,
LinkedListNode<Integer> head2) {

        //Your code goes here

        if(head1==null)

            return head2;

        if(head2==null)

            return head1;

        LinkedListNode<Integer> t1=head1,t2=head2,tail=null,head=null;

        if(t1.data<=t2.data)

        {

            head=t1;

            tail=t1;

            t1=t1.next;

        }

        else

        {

            head=t2;
```

```
          tail=t2;

          t2=t2.next;

       }

       while(t1!=null &&t2!=null)

       {

          if(t1.data<=t2.data)

          {

             tail.next=t1;

             tail=t1;

             t1=t1.next;

          }

          else

          {

             tail.next=t2;

             tail=tail.next;

             t2=t2.next;

          }

       }

       if(t1==null)

          tail.next=t2;

       if(t2==null)

          tail.next=t1;

       return head;

    }


}
```

## Merge Sort LL

Send Feedback

Given a singly linked list of integers, sort it using 'Merge Sort.'

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the sorted singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec

**Sample Input 1 :**

```
1
10 9 8 7 6 5 4 3 -1
```

**Sample Output 1 :**

```
 3 4 5 6 7 8 9 10
```

```
public class Solution {



        public static LinkedListNode<Integer> mergeSort(LinkedListNode<Integer> head) {

                //Your code goes here

    if(head==null)

        return head;

    if(head.next==null)

        return head;

    LinkedListNode<Integer> midNode = findmid(head);

    LinkedListNode<Integer> h2 = midNode.next;

    midNode.next = null;

    LinkedListNode<Integer> part1 = mergeSort(head);

    LinkedListNode<Integer> part2 = mergeSort(h2);

    LinkedListNode<Integer> mergeList = mergeTwoList(part1,part2);

    return mergeList;

        }

  private static LinkedListNode<Integer> findmid(LinkedListNode<Integer>head){

    if(head==null)
```

```
            return head;

        LinkedListNode<Integer>slow=head,fast=head;

        while(fast.next!=null && fast.next.next!=null){

            slow = slow.next;

            fast = fast.next.next;

        }

        return slow;

    }

    public static LinkedListNode<Integer> mergeTwoList(LinkedListNode<Integer> head1,
LinkedListNode<Integer> head2) {

        if(head1==null)

            return head2;

        if(head2==null)

            return head1;

        LinkedListNode<Integer> t1=head1,t2=head2,tail=null,head=null;

        if(t1.data<=t2.data)

        {

            head=t1;

            tail=t1;

            t1=t1.next;

        }

        else

        {

            head=t2;

            tail=t2;

            t2=t2.next;

        }

        while(t1!=null &&t2!=null)

        {
            if(t1.data<=t2.data)

            {

               tail.next=t1;

               tail=t1;

               t1=t1.next;

            }

            else

            {

               tail.next=t2;

               tail=tail.next;

               t2=t2.next;

            }

         }

         if(t1==null)

            tail.next=t2;

         if(t2==null)

            tail.next=t1;

         return head;


      }


   }
```

# TEST

## Find a node in LL (recursive)

Send Feedback

Given a singly linked list of integers and an integer n, find and return the index for the first occurrence of 'n' in the linked list. -1 otherwise.

Follow a recursive approach to solve this.

**Note :**

Assume that the Indexing for the linked list always starts from 0.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'n'.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit:  1sec

**Sample Input 1 :**

```
1
3 4 5 2 6 1 9 -1
100
```

**Sample Output 1 :**

```
-1
```

```
public class Solution {




      public static int findNodeRec(LinkedListNode<Integer> head, int n) {

      //Your code goes here

   if(head==null){

      return -1;

   }

   if(head.data.equals(n)){

      return 0;

   }

   int smallIndex = findNodeRec(head.next,n);

   if(smallIndex ==-1){

      return smallIndex;

   }

   else{
```

```
            return smallIndex+1;

        }

            }



    }
```

## Even after Odd LinkedList

Send Feedback

For a given singly linked list of integers, arrange the elements such that all the even numbers are placed after the odd numbers. The relative order of the odd and even terms should remain unchanged.

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format:**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format:**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec

**Sample Input 1 :**

```
1
1 4 5 2 -1
```

**Sample Output 1 :**

```
1 5 4 2
```

```
public class Solution {



    public static LinkedListNode<Integer> evenAfterOdd(LinkedListNode<Integer> head) {

        //Your code goes here



        if(head==null || head.next==null){

            return head;

        }
```

```
LinkedListNode<Integer>node=head,evenNode=null,oddNode=null,evenHead=null,oddHead=n
ull;
    while(node!=null){
      int data = node.data;
      if(data%2==0){
        if(evenNode==null){
          evenNode = node;
          evenHead = node;
        }
        else{
          evenNode.next = node;
          evenNode = evenNode.next;
        }
      }
      else{
        if(oddNode==null){
          oddNode=node;
          oddHead=node;
        }
        else{
          oddNode.next = node;
          oddNode = oddNode.next;
        }
      }
      node = node.next;
    }
    if(oddHead==null){
      return evenHead;

    }

    else{

      oddNode.next = evenHead;

    }

    if(evenNode!=null){

      evenNode.next=null;

    }

    return oddHead;

         }

}
```

## Delete every N nodes

Send Feedback

You have been given a singly linked list of integers along with two integers, 'M,' and 'N.' Traverse the linked list such that you retain the 'M' nodes, then delete the next 'N' nodes. Continue the same until the end of the linked list.

To put it in other words, in the given linked list, you need to delete N nodes after every M nodes.

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains two integer values 'M,' and 'N,' respectively. A single space will separate them.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= P <= 10^5
Where P is the size of the singly linked list.
0 <= M <= 10^5
0 <= N <= 10^5

Time Limit: 1sec

**Sample Input 1 :**

```
1
1 2 3 4 5 6 7 8 -1
```

```
2 2
```

**Sample Output 1 :**

```
1 2 5 6
```

```
public class Solution {



            public static LinkedListNode<Integer> skipMdeleteN(LinkedListNode<Integer> head, int
M, int N) {

                        //Your code goes here

       if(head==null)

          return head;

       if(M==0)

          return null;

       if(N==0)

          return head;

       LinkedListNode<Integer> curr=head,t;

       int count;

       while(curr!=null)

       {

       for(count=1;count<M && curr!=null;count++)

       {

          curr=curr.next;

       }

       if(curr==null)

          return head;

       t=curr.next;

       for(count=1;count<=N && t!=null;count++)

       {

          t=t.next;

       }
```

```
      curr.next=t;

      curr=t;}

      return head;

         }

}
```

## Swap two Nodes of LL

Send Feedback

You have been given a singly linked list of integers along with two integers, 'i,' and 'j.' Swap the nodes that are present at the 'i-th' and 'j-th' positions.

**Note :**

Remember, the nodes themselves must be swapped and not the datas.

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains two integer values 'i,' and 'j,' respectively. A single space will separate them.

**Remember/consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.
0 <= i < M
0 <= j < M

Time Limit: 1sec

**Sample Input 1 :**

```
1
3 4 5 2 6 1 9 -1
3 4
```

**Sample Output 1 :**

```
3 4 5 6 2 1 9
```

```
public class Solution {



	public static LinkedListNode<Integer> swapNodes(LinkedListNode<Integer> head, int i, int j) {

		//Your code goes here

     LinkedListNode<Integer> temp1=head;
```

```
        LinkedListNode<Integer> node1=head;

        LinkedListNode<Integer> node2=head;



        for(int a=0;a<i;a++)

            node1=node1.next;

         for(int b=0;b<j;b++)

                        node2=node2.next;

        int temp=node1.data;

        node1.data=node2.data;

        node2.data=temp;

        return temp1;

            }



}
```

## kReverse

Send Feedback

Given a singly linked list of integers, reverse the nodes of the linked list 'k' at a time and return its modified list.

'k' is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of 'k,' then left-out nodes, in the end, should be reversed as well.

**Example :**

```
Given this linked list: 1 -> 2 -> 3 -> 4 -> 5

For k = 2, you should return: 2 -> 1 -> 4 -> 3 -> 5

For k = 3, you should return: 3 -> 2 -> 1 -> 5 -> 4
```

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'k'.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a separate line.

**Constraints :**

```
1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.
0 <= k <= M

Time Limit:  1sec
```

**Sample Input 1 :**

```
1
1 2 3 4 5 6 7 8 9 10 -1
4
```

**Sample Output 1 :**

```
4 3 2 1 8 7 6 5 10 9
```

```
public class Solution {



            public static LinkedListNode<Integer> kReverse(LinkedListNode<Integer> head, int k) {

                        //Your code goes here

        if(head==null)

            return head;

        if(head.next==null)

            return head;

        if(k==0)

            return head;

        LinkedListNode<Integer> h1=head,h2,t1=head;

        int count=1;

     while(count<k && t1.next!=null)

     {

         t1=t1.next;

         count++;

     }



            h2=t1.next;

            t1.next=null;
        DoubleNode ans=reversePart(h1);

        LinkedListNode<Integer> secondHead=kReverse(h2,k);

        ans.tail.next=secondHead;

        return ans.head;


    }

    private static DoubleNode reversePart(LinkedListNode<Integer> head)

    {

        if(head==null || head.next==null)

        { DoubleNode ans=new DoubleNode();

        ans.head=head;

        ans.tail=head;

        return ans;}


        DoubleNode ans=reversePart(head.next);

        ans.tail.next=head;

        head.next=null;

        ans.tail=ans.tail.next;

        return ans;

    }


}

class DoubleNode{

    LinkedListNode<Integer> head;

    LinkedListNode<Integer> tail;

}
```

## Bubble Sort (Iterative) LinkedList

Send Feedback

Given a singly linked list of integers, sort it using 'Bubble Sort.'

**Note :**

No need to print the list, it has already been taken care. Only return the new head to the list.

**Input format :**

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

**Remember/Consider :**

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

**Output format :**

For each test case/query, print the elements of the sorted singly linked list.

Output for every test case will be printed in a seperate line.

**Constraints :**

0 <= M <= 10^3

Where M is the size of the singly linked list.

Time Limit: 1sec

**Sample Input 1 :**

10 9 8 7 6 5 4 3 -1

**Sample Output 1 :**

 3 4 5 6 7 8 9 10

```
public class Solution {



	public static LinkedListNode<Integer> bubbleSort(LinkedListNode<Integer> head) {

		//Your code goes

  LinkedListNode<Integer> current = head, index = null;

   int temp;



   if(head == null) {

       return head;

   }

   else

   {

       while(current != null)

       {

           //Node index will point to node next to current

           index = current.next;
```

```
                    while(index != null)
                {
                  //If current node's data is greater than index's node data, swap the data between
them
                  if(current.data > index.data)
                  {
                     temp = current.data;
                     current.data = index.data;
                     index.data = temp;
                  }
                  index = index.next;
                        }
                        current = current.next;
            }

        }  return head;

            }
}
```