» Skills Enhancement

» Placement Readiness

» Proficiency Diagnostics

# check if a singly linked list is palindrome

Given a singly linked list of integers, write a program that print YES if the given list is palindrome, else NO.

**Input :**

The first line of input contains length L of linked list and last line contains the data of the linked list.

**Output :**

The first line of input contains string, that Prints YES if the linked list is palindrome , if not print NO.

**Constraints:**

$1 <= L <= 100$

$1 <= N <= 10^3$

# Example:

**Input**:

5

1 2 2 2 1

**Output**:

YES

**Input**:

3

1 2 3

**Output**:

NO

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node* next;
};
void create(struct Node** head, int data)
{
        struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));

        newNode->data = data;
        newNode->next = *head;
        *head = newNode;
}
int checkPalindrome(struct Node** left, struct
Node* right)
{
        if (right == NULL)
                return 1;
```

```
int res = checkPalindrome(left, right->next)
&&
 ((*left)->data == right->data);
(*left) = (*left)->next;

        return res;
}

int checkPalin(struct Node* head)
{
        return checkPalindrome(&head,
head);
}

int main(void)
{
struct Node* head = NULL;
        int n,data;
        printf("Enter No Of Nodes\n");
        scanf("%d",&n);
```

```
while(n--)
{
printf("enter node data\n");
scanf("%d", &data);
create(&head, data);
}

   if (checkPalin(head))
                printf("YES");
     else
            printf("NO");

return 0;
}
```

# Rotate a Linked List

Given a singly linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer.

**Input:**

First line of input contains length L of linked list and next line contains the data of the linked list. The last line contains the node to be rotated.

**Output:**

Print the linked list after rotating the list.

- **Constraints:**

  $1 <= L <= 100$

  $1 <= N <= 10^3$

  $1 <= R <= 100$

Example:

**Input**:

6

10 20 30 40 50 60

4

**Output**:

50 60 10 20 30 40

**Explanation:**

if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
void rotate(struct Node **head_ref, int k)
{
    if (k == 0)
      return;
   struct Node* current = *head_ref;
   int count = 1;
   while (count < k && current != NULL)
   {
      current = current->next;
      count++;
   }
   if (current == NULL)
       return;
   struct Node *kthNode = current;
   while (current->next != NULL)
        current = current->next;
        current->next = *head_ref;
       *head_ref = kthNode->next;
        kthNode->next = NULL;
}
```

```
void push (struct Node** head_ref, int new_data)
{
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}
```

```
int main(void)
{
   struct Node* head = NULL;
   int i, a;
   for (i = 60; i > 0; i -= 10)
      push(&head, i);

   printf("Given linked list \n");
   printList(head);
   printf("\nGive the node to be rotated\n");
            scanf("%d", &a);
   rotate(&head, a);

   printf("\nRotated Linked list \n");
   printList(head);

   return (0);
}
```

# Compare two linked lists

You're given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. The lists are equal only if they have the same number of nodes and corresponding nodes contain the same data. Either head pointer given may be null meaning that the corresponding list is empty.

**Input Format**

The first line contains an integer , denoting the number of elements in the first linked list.
The next  lines contain an integer each, denoting the elements of the first linked list.
The next line contains an integer , denoting the number of elements in the second linked list.
The next  lines contain an integer each, denoting the elements of the second linked list.

**Output :**

The first line of input contains string,
Print YES if both the linked lists are same , if not print NO.

EXAMPLE:

**Input**:

4

1 2 3 4

4

1 2 3 4

**Output**:

YES

**Input**:

2

5 6

3

5 6 7

**Output**:

NO

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
      int data;
            struct node *next;
};
struct node *head1 = NULL;
struct node *head2 = NULL;

int compare_ll(struct node *head1, struct node
*head2)
{
if(head1 == NULL )
return 0;
if(head2 == NULL )
return 0;
```

```
while (head1 && head2)
{
 if(head1 -> data == head2 -
> data)
    {
     head1 = head1 -> next ;
     head2 = head2 -> next ;
    }
   else
   {
    return 0;
   }
}
```

```
if(head1 == NULL && head2 == NULL)
return 1;
else
return 0;
}
void display(struct node *temp)
{
        while(temp!=NULL)
        {
        printf("\n%d ",temp->data);
    temp=temp->next;
        }
}

int main()
{
    struct node *a = (struct node*)malloc(sizeof(struct node));
    struct node *b = (struct node*)malloc(sizeof(struct node));
    struct node *c = (struct node*)malloc(sizeof(struct node));
    struct node *d = (struct node*)malloc(sizeof(struct node));
```

```
struct node *e = (struct node*)malloc(sizeof(struct node));
 struct node *f = (struct node*)malloc(sizeof(struct node));
                    head1=a;
          head2=b;                                  f->data=3;
          a->data=1;                                f->next=NULL;
          a->next=c;
                                                      printf("\nInput: Linked List 1");
          b->data=1;                                        display(head1);
          b->next=d;
                                                    printf("\nInput: Linked List 2");
          c->data=2;                                        display(head2);
          c->next=e;

          d->data=2;                                        if(compare_ll(head1,head2)
          d->next=f;                              )
                                                            printf("\nYES");
          e->data=4;                                        else
          e->next=NULL;                                     printf("\nNO");
                                                            return 0;
                                                  }
```

# Merge two sorted linked lists

Write a SortedMerge() function that takes two lists, each of which is sorted in increasing order, and merges the two together into one list which is in increasing order.

## Input :

Do not read any input from console.
The SortedMerge() method has two parameters,  and , which are the non-null head Nodes of two separate linked lists that are guaranteed to converge.

## Output :

The output list should be made by splicing
together the nodes of the two lists.

Example:
2 3 20
5 10 15
**Output :**
Merged Linked List is: 2 3 5 10 15 20

**Explanation:**
For example if the first linked list a is 5->10->15 and the other linked list b is 2->3->20, then SortedMerge() should return a pointer to the head node of the merged list 2->3->5->10->15->20.

```
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
struct Node
{
   int data;
   struct Node* next;
};
void MoveNode(struct Node** destRef, struct
Node** sourceRef);
struct Node* SortedMerge(struct Node* a, struct
Node* b)
{
   struct Node dummy;
   struct Node* tail = &dummy;
   dummy.next = NULL;
  while (1)
   {
      if (a == NULL)
      {
         tail->next = b;
         break;
      }
else if (b == NULL)
      {
         tail->next = a;
         break;
      }
      if (a->data <= b-
>data)
         MoveNode(&(tail-
>next), &a);
      else
         MoveNode(&(tail-
>next), &b);

      tail = tail->next;
   }
   return(dummy.next);
}
```

```
void MoveNode(struct Node** destRef, struct
Node** sourceRef)
{
   struct Node* newNode = *sourceRef;
   assert(newNode != NULL);
   *sourceRef = newNode->next;
   newNode->next = *destRef;
   *destRef = newNode;
}
void push(struct Node** head_ref, int new_data)
{
   struct Node* new_node =
      (struct Node*) malloc(sizeof(struct Node));
   new_node->data  = new_data;
   new_node->next = (*head_ref);
   (*head_ref)    = new_node;
}
```

```
void printList(struct Node
*node)
{
   while (node!=NULL)
   {
      printf("%d ", node->data);
      node = node->next;
   }
}
```

```
int main()
{
    struct Node* res = NULL;
    struct Node* a = NULL;
    struct Node* b = NULL;
    push(&a, 15);
    push(&a, 10);
    push(&a, 5);
    push(&b, 20);
    push(&b, 3);
    push(&b, 2);
    res = SortedMerge(a, b);

    printf("Merged Linked List is: \n");
    printList(res);

    return 0;
}
```

# Merge a linked list into another linked list at alternate positions

Given two linked lists, insert nodes of second list into first list at alternate positions of the list.

**Input :**
Do not read any input from console.
The Merge() method is use for merging the two linked list at alternate positions.

**Output :**
The output list should be made by splicing together the nodes of the two lists.

**Example :**
Input
Linked List 1 : 1 3 5 7
Linked List 2 :  2 4 6

Output
1 2 3 4 5 6 7

**Explanation:**
For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
      int data;
            struct node *next;
};

struct node *head1 = NULL;
struct node *head2 = NULL;
void merge( )
{
   struct node *temp1 = head1;
            struct node *temp2 = head2;
            struct node *holder1 = NULL;
            struct node *holder2 = NULL;
            while(temp1!=NULL && temp2!=NULL)

            {
                        holder1=temp1->next;
                        temp1->next=temp2;
                        if(holder1!=NULL){
```

```
holder2=temp2->next;
                    temp2->next=holder1;
                            }

                       temp1=holder1;
                       temp2=holder2;
            }
}
void display(struct node *temp)
{
            while(temp!=NULL)
            {
            printf("\n%d ",temp->data);
      temp=temp->next;
            }
}
int main()
{
      struct node *a = (struct node*)malloc(sizeof(struct node));
            struct node *b = (struct node*)malloc(sizeof(struct node));
```

```
struct node *c = (struct node*)malloc(sizeof(struct node));
struct node *d = (struct node*)malloc(sizeof(struct node));
struct node *e = (struct node*)malloc(sizeof(struct node));
struct node *f = (struct node*)malloc(sizeof(struct node));
struct node *g = (struct node*)malloc(sizeof(struct node));
head1=a;
head2=b;
a->data=1;                                  g->data=7;
a->next=c;                                  g->next=NULL;
                                            printf("\nBefore Merging");
b->data=2;                                  printf("\nInput: Linked List 1");
b->next=d;                                  display(head1);

c->data=3;                                  printf("\nInput: Linked List 2");
c->next=e;                                  display(head2);

d->data=4;                                  merge();
d->next=f;                                  printf("\nAfter Merging");
                                            printf("\nOutput: Linked List 3");
e->data=5;                                  display(head1);
e->next=g;                                  return 0;
                                    }
f->data=6;
f->next=NULL;
```

# MERGE TWO SORTED LINKED LIST

## Input Format

You have to complete the SinglyLinkedListNode MergeLists(SinglyLinkedListNode headA, SinglyLinkedListNode headB) method which takes two arguments - the heads of the two sorted linked lists to merge. You should NOT read any input from stdin/console.

The input is handled by the code in the editor and the format is as follows:

The first line contains an integer , denoting the number of test cases.
The format for each test case is as follows:

The first line contains an integer , denoting the length of the first linked list.
The next  lines contain an integer each, denoting the elements of the linked list.
The next line contains an integer , denoting the length of the second linked list.
The next  lines contain an integer each, denoting the elements of the second linked list.
Output Format

Change the next pointer of individual nodes so that nodes from both lists are merged into a single list. Then return the head of this merged list. Do NOT print anything to stdout/console.

**The output is handled by the editor and the format is as follows:**
For each test case, print in a new line, the linked list after merging them separated by spaces.

Sample Input

1
3
1
2
3
2
3
4
Sample Output

1 2 3 3 4

```
#include<stdio.h>
  SinglyLinkedListNode {
      int data;
      SinglyLinkedListNode* next;
  };
SinglyLinkedListNode* mergeLists(SinglyLinkedListNode* head1,
SinglyLinkedListNode* head2) {
struct SinglyLinkedListNode* start=(struct
SinglyLinkedListNode*)malloc(sizeof(struct SinglyLinkedListNode));

int count=0;
struct SinglyLinkedListNode* temp= (struct
SinglyLinkedListNode*)malloc(sizeof(struct SinglyLinkedListNode));
if (head1 == NULL)return head2;
else if (head2 == NULL)return head1;
else if (head1 == NULL && head2 == NULL) return NULL;
else{
while ((head1 != NULL) && (head2 != NULL)){
   if((head1 != NULL) && (head2 != NULL) && (head1->data < head2->data) ){
       if (count==0) start=head1;
       while((head1 != NULL) && (head1->data < head2->data)){
       printf("%d\n",head1->data) ;
```

```
temp=head1;
      head1=head1->next;
      }

      if ((head1 != NULL) && (head2 != NULL)){
         if (head1->data == head2->data){
            temp=head1;
            head1=head1->next;
            temp->next=head2;
         }
      }
      else{
         count=count+1;
      temp->next=head2;
      temp=NULL;
      }
   }
   if((head1 != NULL) && (head2 != NULL) && (head2->data < head1->data)) {
      if (count==0) start=head2;
      while((head2 != NULL) && (head2->data < head1->data) ){

         temp=head2;
```

```
 head2=head2->next;

       }
       count=count+1;
       temp->next=head1;
       temp=NULL;
   }

   if((head1 != NULL) && (head2 != NULL) && (head1->data == head2-
>data)){
       printf("Hi");
       break;
   }

}
return start;

}
}
```

# Inserting a Node Into a Sorted Doubly Linked List

## Input Format

The first line contains an integer , the number of test cases.
Each of the test case is in the following format:
The first line contains an integer , the number of elements in the linked list.
Each of the next  lines contains an integer, the data for each node of the linked list.
The last line contains an integer  which needs to be inserted into the sorted doubly-linked list.
Constraints

## Output Format

Do not print anything to stdout. Your method must return a reference to the  of the same list that was passed to it as a parameter.

The ouput is handled by the code in the editor and is as follows:
For each test case, print the elements of the sorted doubly-linked list separated by spaces on a new line.

```
#include<stdio.h>
  DoublyLinkedListNode
{
     int data;
     DoublyLinkedListNode* next;
     DoublyLinkedListNode* prev;
  };
DoublyLinkedListNode* sortedInsert(DoublyLinkedListNode* head, int data)
{
      DoublyLinkedListNode *newnode=create_doubly_linked_list_node(data);

   if(head==NULL)
      head=newnode;
   else{
        DoublyLinkedListNode *temp=head;
        if(temp->data>=newnode->data)
        {
           newnode->next=temp;
           temp->prev=newnode;
           head=newnode;
        }
        else{
```

```
 while(temp!=NULL&&temp->next!=NULL)
            {
                if(temp->data<newnode->data&&temp->next->data>newnode->data)
                {
                   newnode->next=temp->next;
                   temp->next->prev=newnode;
                   temp->next=newnode;
                   newnode->prev=temp;
                   break;
                }
                temp=temp->next;
            }
            if(temp->next==NULL)
            {
               temp->next=newnode;
               newnode->prev=temp;
                                                    }
                                                        }
                                                     }
                                                   return head;
                                                  }
```

# Find Merge Point of Two Lists

## Input Format

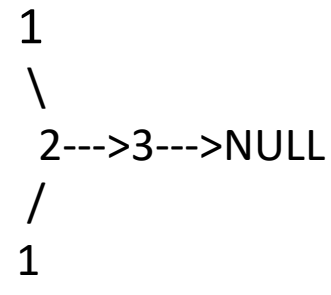
Do not read any input from stdin/console.
The findMergeNode(SinglyLinkedListNode,SinglyLinkedListNode) method has two parameters,  and , which are the non-null head Nodes of two separate linked lists that are guaranteed to converge.
Do not write any output to stdout/console.

Each Node has a data field containing an integer. Return the integer data for the Node where the two lists merge.

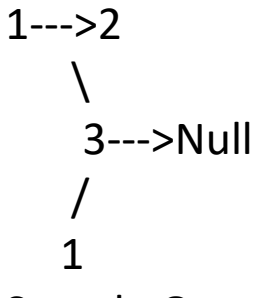
## Sample Input

The diagrams below are graphical representations of the lists that input Nodes and  are connected to. Recall that this is a method-only challenge; the method only has initial visibility to those  Nodes and must explore the rest of the Nodes using some algorithm of your own design.

## Test Case 0




Test Case 1




Sample Output

2
3

```
#include<stdio.h>
  SinglyLinkedListNode
{
     int data;
    SinglyLinkedListNode* next;
  };
int findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2)
{
struct SinglyLinkedListNode *list1 =head1;
   struct SinglyLinkedListNode *list2=head2;
   int count1=1, count2=1;
   while(list1->next!=NULL)
   {
      list1=list1->next;
      count1++;
   }
   while(list2->next!=NULL)
   {
      list2=list2->next;
      count2++;
   }
   if(count1>=count2)
```

```
{
    int count=count1-count2;
      while(count!=0)
      {
         head1=head1->next;
         count--;
      }
  }
         else if(count2>count1)
  {
      int count=count2-count1;
      while(count!=0)
      {
         head2=head2->next;
         count--;
      }
  }
  while(head1!=NULL && head2!=NULL)
  {

     if(head1->next==head2->next)
      return (head2->data);
  }
}
```

# Reverse a doubly linked list

## Input Format

The first line contains an integer , the number of test cases.
Each test case is of the following format:
The first line contains an integer , the number of elements in the linked list.
The next  lines contain an integer each denoting an element of the linked list.
Constraints

## Output Format

Return a reference to the head of your reversed list. The provided code will print the reverse array as a one line of space-separated integers for each test case.

## Sample Input

1
4
1
2
3
4

## Sample Output

4 3 2 1

```
 #include<stdio.h>
 DoublyLinkedListNode
{
    int data;
    DoublyLinkedListNode* next;
    DoublyLinkedListNode* prev;
 };
DoublyLinkedListNode* reverse(DoublyLinkedListNode* head) {
   DoublyLinkedListNode* p=NULL;
   DoublyLinkedListNode* q=NULL;
   DoublyLinkedListNode* headNew=NULL;
   p=head;
   while(p!=NULL){
      if(p==head){

headNew=(DoublyLinkedListNode*)malloc(sizeof(DoublyLinkedListNode));
         headNew->data=head->data;
         p=p->next;
      }
      else{
```

```
 DoublyLinkedListNode* newNode
=(DoublyLinkedList*)malloc(sizeof(DoublyLinkedListNode));
          newNode->data= p->data;
          headNew->prev=newNode;
          newNode->next=headNew;
          headNew=newNode;
          p=p->next;
       }
    }
    return headNew;

}
```

# Get Node Value

## Input Format

You have to complete the int getNode(SinglyLinkedListNode* head, int positionFromTail) method which takes two arguments - the head of the linked list and the position of the node from the tail. positionFromTail will be at least 0 and less than the number of nodes in the list. You should NOT read any input from stdin/console.

The first line will contain an integer , the number of test cases.
Each test case has the following format:
The first line contains an integer , the number of elements in the linked list.
The next  lines contains, an element each denoting the element of the linked list.
The last line contains an integer  denoting the position from the tail, whose value needs to be found out and returned.

## Constraints

, where  is the  element of the linked list.

Output Format

Find the node at the given position counting backwards from the tail. Then return the data contained in this node. Do NOT print anything to stdout/console.

The code in the editor handles output.

For each test case, print the value of the node, each in a new line.

## Sample Input

2
1
1
0
3
3
2
1
2

## Sample Output

1
3

```
#include<stdio.h>
 SinglyLinkedListNode
{
     int data;
     SinglyLinkedListNode* next;
 };
int getNode(SinglyLinkedListNode* head, int positionFromTail)
{
   SinglyLinkedListNode *p=head;
   int total=1,r;
   while(p->next!=NULL){
      p=p->next;
      total++;
   }
   r=total-positionFromTail;
   p=head;
   for(int i=1;i<=r;i++)
{
      p=p->next;
   }
   int result=p->data;
return result;

}
```

# Remove Friends

After getting her PhD, Christie has become a celebrity at her university, and her facebook profile is full of friend requests. Being the nice girl she is, Christie has accepted all the requests.
Now Kuldeep is jealous of all the attention she is getting from other guys, so he asks her to delete some of the guys from her friend list.
To avoid a 'scene', Christie decides to remove some friends from her friend list, since she knows the popularity of each of the friend she has, she uses the following algorithm to delete a friend.

```
Algorithm Delete(Friend):
   DeleteFriend=false
   for i = 1 to Friend.length-1
       if (Friend[i].popularity < Friend[i+1].popularity)
          delete i th friend
          DeleteFriend=true
          break
   if(DeleteFriend == false)
      delete the last friend
```

## Input:

First line contains T number of test cases. First line of each test case contains N, the number of friends Christie currently has and K ,the number of friends Christie decides to delete. Next lines contains popularity of her friends separated by space.

## Output:

For each test case print N-K numbers which represent popularity of Christie friend's after deleting K friends.

## Constraints

1<=T<=1000
1<=N<=100000
0<=K< N
0<=popularity_of_friend<=100

**NOTE:**

Order of friends after deleting exactly K friends should be maintained as given in input.

**SAMPLE INPUT**

```
3
3 1
3 100 1
5 2
19 12 3 4 17
5 3
23 45 11 77 18
```

**SAMPLE OUTPUT**

```
#include<stdio.h>
#include<stdlib.h>

struct Node{
        int data;
        struct Node* next;
};
struct Node* head;
struct Node* end=NULL;

void Insert(int data)
{       struct Node* temp=(struct Node*)malloc(sizeof(struct Node*));
        temp->data=data;
        temp->next=NULL;
        if(head==NULL)
        {
                end=temp;
                head=temp;
                return;
        }
        end->next=temp;
        end=temp;
```

```
}
void Delete(struct Node* p)
{
        struct Node* temp=p->next;
        p->data=temp->data;
        p->next=temp->next;
        free(temp);

}
 void Print(struct Node* p)
{
        if(p==NULL)
        {
                return;
        }
        printf("%d ",p->data);
        Print(p->next);
}
/*int len(struct Node* p)
{
        struct Node* temp=head;
        int count=1;
```

```
        while(temp!=NULL)
        {
                count++;
                temp=temp->next;
        }
        return count;
}*/
void DeleteFriend(struct Node* head,int k)
{       int i,ln,dltfrnd;
        while(k>0)
        {
                dltfrnd=0;
                struct Node* current=head;
                struct Node* next=current->next;
                while(next)
                {
                        if(current->data < next->data)
                        {
                                dltfrnd=1;
                                k--;
                                Delete(current);
                                break;
```

```
                        }
                        current=next;
                        next=current->next;
                }
                if(dltfrnd==0)
                {
                        Delete(next);
                        k--;
                }
        }
}
 main()
{
        int t,i;
        scanf("%d",&t);
        for(i=0;i<t;i++)
        {       head=NULL;
                int n,k,j,x;
                scanf("%d %d",&n,&k);
                for(j=0;j<n;j++)
```

```
        {
                scanf("%d",&x);
                Insert(x);
            }
            DeleteFriend(head,k);
            Print(head);
            printf("\n");

        }
}
```

# Linked List Length Even or Odd

**User Task:**

Since this is a functional problem you don't have to worry about input, you just have to  complete the function isLengthEvenOrOdd().

**Constraints:**

1 <= T <= 100

1 <= N <= 103

1 <= A[i] <= 103

**Example:**

Input:

2

3

9 4 3

6

12 52 10 47 95 0

**Output:**

Odd

Even

```
#include<stdio.h>
struct Node
{
        int data;
        struct Node* next;
};
int isLengthEvenOrOdd(struct Node* head)
{
     int cnt;
  struct Node *p;
  p=head;
  while(p->next!=NULL)
  {
    cnt++;
    p=p->next;
   }
 if((cnt+1)%2==0)
    return 0;
    else
   return 1;
 }
```

# Count Pairs whose sum is equal to X in LL

Given two linked list of size N1 and N2 respectively of distinct elements, your task is to complete the function countPairs(), which returns the count of all pairs from both lists whose sum is equal to the given value X.

## Input:

The function takes three arguments as input, reference pointer to the head of the two linked list (head1 and head2), and an variable X.
There will be T test cases and for each test case the function will be called separately.

## Output:

For each test case the function should return the count of all the pairs from both lists whose sum is equal to the given value X.

**Constraints:**

1<=T<=100

1<=N1,N2<=1000

1<=X<=10000

**Example:**

**Input:**

2

6

1 2 3 4 5 6

3

11 12 13

15

4

7 5 1 3

4

3 5 2 8

10

**Output:**

3

2

```
#include<stdio.h>
struct Node
{
  int data;
  struct Node* next;
};
int countPairs(struct Node* head1,
struct Node* head2,          int x)
{
   int cnt=0;
     while(head1)
  {
      int temp=x-head1->data;
       struct Node *p=head2;
        while(p)
        {
          if(p->data==temp)
              {
              cnt+=1;
                 break;
              }
        p=p->next;
        }
         head1=head1->next;
    }
    return cnt;
}
```

# Quick Sort on Linked List

Sort the given Linked List using quicksort. which takes O(n^2) time in worst case and O(nLogn) in average and best cases, otherwise you may get TLE.

## Input:

In this problem, method takes 1 argument: address of the head of the linked list. The function should not read any input from stdin/console.
The struct Node has a data part which stores the data and a next pointer which points to the next element of the linked list.
There are multiple test cases. For each test case, this method will be called individually.

## Output:

Set *headRef to head of resultant linked list.

## Constraints:

1<=T<=100
1<=N<=200

**Note:** If you use "Test" or "Expected Output Button" use below example format

## Example:

## Input

2
3
1 6 2
4
1 9 3 8

## Output

1 2 6
1 3 8 9

```
#include<stdio.h>
struct node
{
   int data;
 struct node *next;
};
void swap(struct node* n1, struct
node* n2)
{
   int temp = n1->data;
   n1->data = n2->data;
  n2->data = temp;
 return;
}
struct node* quick(struct node*
head)
{
   struct node* last = head;
  if(head==NULL || head-
>next==NULL)
{
      return head;
}
  while(last->next!=NULL)
{
 last = last->next;
  }
  struct node* thead = head;
   struct node* i = head;
  struct node* fl = NULL;
 while(thead!=last)
{
     if(thead->data<last->data)
{
    swap(thead, i);
       fl = i;
      i = i->next;
   }
     thead = thead->next;
  }
  swap(last,i);
   if(fl==NULL)
{
      quickSort(&(i->next));
```

```
 return i;
    }
  fl->next=NULL;
   quickSort(&head);
    quickSort(&i);
   thead = head;
 while(head->next!=NULL)
       head = head->next;
 head->next = i;
  return thead;
}

void quickSort(struct node **headRef)
{
   struct node* head = *headRef;
    if(head==NULL || head->next==NULL)
       return;
    *headRef = quick(head);
  return;
}
```

# Occurrence of an integer in a Linked List

Given a singly linked list and a key, count number of occurrences of given key in linked list. For example, if given linked list is 1->2->1->2->1->3->1 and given key is 1, then output should be 4.

## Input:

You have to complete the method which takes two argument: the head of the linked list and int k. You should not read any input from stdin/console.
The struct Node has a data part which stores the data and a next pointer which points to the next element of the linked list.
There are multiple test cases. For each test case, this method will be called individually.

## Output:

You have to count a number of occurrences of given key in linked list and return the count value.

**Note:** If you use "Test" or "Expected Output Button" use below example format

**Example:**

**Input:**

1
8
1 2 2 4 5 6 7 8
2

**Output:**

2

```
#include<stdio.h>
struct node
{
    int data;
    struct node *next;
 };
int count(struct node* head, int search_for)
{
int c=0;
while(head->next)
{
    if(head->data==search_for)
   c++;
 head=head->next;
   }
if(head->data==search_for)
c++;
return c;
}
```

# Linked List Matrix

Given a Matrix mat of N*N size, the task is to complete the function construct LinkedMatrix(), that constructs a 2D linked list representation of the given matrix.

**Input :** 2D matrix
1 2 3
4 5 6
7 8 9

**Output :**
1 -> 2 -> 3 -> NULL
| | |
v v v
4 -> 5 -> 6 -> NULL
| | |
v v v
7 -> 8 -> 9 -> NULL
| | |
v v v
NULL NULL NULL

## Input:

The fuction takes 2 argument as input, first the 2D matrix mat[][] and second an integer variable N, denoting the size of the matrix.
There will be T test cases and for each test case the function will be called separately.

## Output:

The function must return the reference pointer to the head of the linked list.

## Constraints:

1<=T<=100
1<=N<=150

## Example:

**Input:**

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

**Output:**

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

```
struct Node
{
        int data;
Node* right, *down;
};
Node *createNode(int x)
{
struct Node *temp = new Node;
temp->data = x;
temp->right = NULL;
temp->down = NULL;
return temp;
}
Node* construct(int mat[MAX][MAX],int i,int j,int n)
{
  if(i>n-1 || j>n-1)
 return NULL;
   Node* temp=createNode(mat[i][j]);
  temp->right=construct(mat,i,j+1,n);
  temp->down=construct(mat,i+1,j,n);
 return temp;
```

```
}
Node* constructLinkedMatrix(int mat[MAX][MAX], int n)
{
  return construct(mat,0,0,n);
}
```

Given Pointer/Reference to the head of a linked list, task is to Sort the given linked list using Merge Sort.

You need to complete the function splitList() and mergeList(), which will be called by the function mergeSort().

**Note:** If the length of linked list is odd, then the extra node should go in the first list while splitting.

## Input:

There will be multiple test cases, for each test case function mergeSort() will be called separately. The only input to the function is the pointer/reference to the head of the linked list.

## Output:

For each test, in a new line, print the sorted linked list.

**Constraints:**

1 <= T <= 100

1 <= N <= 105

**Example:**

**Input:**

2

5

3 5 2 4 1

3

9 15 0

**Output:**

1 2 3 4 5

0 9 15

```
#include<stdio.h>
struct node
{
int data;
struct node* next;
};
void splitList(struct node* source, struct node** frontRef, struct node**
backRef)
{
   node *fast,*slow;
   slow=source;
   fast=source->next;
  while(fast!=NULL)
  {
     fast=fast->next;
      if(fast!=NULL)
      {
         slow=slow->next;
           fast=fast->next;
  }
       }
  *frontRef=source;
```

```
*backRef=slow->next;
 slow->next=NULL;
}
struct node* mergeList(struct node* a, struct node* b)
{
    node* result=NULL;
    if(a==NULL)
 return b;
  if(b==NULL)
   return a;
    if(a->data<=b->data)
    {
       result=a;
     result->next=mergeList(a->next,b);
   }
  else
  {
      result=b;
      result->next=mergeList(a,b->next);
  }
  return result;
}
```