Q: **Reorder List**

Given a singly linked list: $A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_{n-1} \rightarrow A_n$, reorder it to: $A_0 \rightarrow A_n \rightarrow A_1 \rightarrow A_{n-1} \rightarrow A_2 \rightarrow A_{n-2} \rightarrow \ldots$
For example: Given 1->2->3->4->5 its reorder is 1->5->2->4->3.

Note: It is recommended do this in-place without altering the nodes' values.

Example 1:

```
Input:
LinkedList: 1->2->3
Output: 1 3 2
```

Example 2:

```
Input:
LinkedList: 1->7->3->4
Output: 1 4 7 3.
```

Approach:1

1. This a Brute force approach, basically we will iterate one node from head to the last node.

2. Now we will point the head node to this last node.

3. Again we will repeat this step and point second node to second last node.

4. Similarly we will repeat this step until we reach the middle most node and then point it’s next to NULL.

Time Complexity: O(N^2).

Approach:2

1. This also similar to previous approach, just in this method we will not travel again and again to last but we will store the addresses of all nodes in a vector.

2. therefore for pointing head to last node we can directly access last node’s address from the vector.

3. And similarly second node can be pointed to second last node and so on..

Time Complexity: O(n)

Space Complexity: O(n)

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

/* Linked list Node */

struct Node {

  int data;

  struct Node* next;


  Node(int x) {

    data = x;

    next = NULL;

  }

};


void reorderList(struct Node* head);


/* Function to create a new Node with given data */

struct Node* newNode(int data) {

  struct Node* new_Node = new Node(data);

  new_Node->data = data;
```

```
    new_Node->next = NULL;

    return new_Node;

}


void printList(struct Node* Node) {

    while (Node != NULL) {

        printf("%d ", Node->data);

        Node = Node->next;

    }

    printf("\n");

}


void freeList(struct Node* head) {

    struct Node* temp;

    while (head != NULL) {


        temp = head;

        head = head->next;

        free(temp);

    }

}


int main(void) {

    int t, n, m, i, x;
    cin >> t;

    while (t--) {

        struct Node* temp, *head;

        cin >> n;

        cin >> x;

        head = new Node(x);

        temp = head;

        for (i = 0; i < n - 1; i++) {

            cin >> x;

            temp->next = new Node(x);

            temp = temp->next;

        }


        reorderList(head);

        printList(head);

        freeList(head);

    }

    return 0;

}

// } Driver Code Ends




/* Following is the Linked list node structure */


/*
```

```
struct Node

{

   int data;

   struct Node* next;



   Node(int x){

      data = x;

      next = NULL;

   }

};

*/

#include<bits/stdc++.h>

Node* reverseList(Node *head)

{

   if(head==NULL || head->next==NULL){

      return head;

   }

   Node *t=reverseList(head->next);

   head->next->next=head;

   head->next=NULL;

   return t;

}
```

void reorderList(Node* head) {

   // Your code here

   struct Node*p;

```
p=head;


vector<struct Node*> v;

int l,i;


while(p!=NULL)

{

   v.push_back(p);

   p=p->next;

}

l=v.size();

p=head;


for(i=0;i<l/2;i++)

{

   if(i!=0)

   {

      p->next=v[i];

      p=p->next;

   }

   p->next=v[l-i-1];

   p=p->next;

}


if(l%2==1)
```

```
  {
      p->next=v[i];
      p=p->next;
  }
  p->next=NULL;
}
```

Approach-3:

1. We will divide the given link list into 2 parts from middle using Floyd cycle Algorithm.

2. After that we will reverse the second part.

3. And then we will interchange the links. Point first node of first part to first node of second part and then point this node to second node of first part and similarly repeat this procedure until both node reached to NULL.

Time Complexity: O(n)

Space Complexity: O(1)

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
/* Linked list Node */
struct Node {
  int data;
  struct Node* next;
```

```
    Node(int x) {
        data = x;
        next = NULL;
    }
};

void reorderList(struct Node* head);

/* Function to create a new Node with given data */
struct Node* newNode(int data) {
    struct Node* new_Node = new Node(data);
    new_Node->data = data;
    new_Node->next = NULL;
    return new_Node;
}

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

void freeList(struct Node* head) {
    struct Node* temp;

    while (head != NULL) {


        temp = head;

        head = head->next;

        free(temp);

    }

}


int main(void) {

    int t, n, m, i, x;


    cin >> t;

    while (t--) {

        struct Node* temp, *head;

        cin >> n;

        cin >> x;

        head = new Node(x);

        temp = head;

        for (i = 0; i < n - 1; i++) {

            cin >> x;

            temp->next = new Node(x);

            temp = temp->next;

        }
        reorderList(head);

        printList(head);

        freeList(head);

    }

    return 0;

}

// } Driver Code Ends




/* Following is the Linked list node structure */


/*

struct Node

{

    int data;

    struct Node* next;


    Node(int x){

        data = x;

        next = NULL;

    }

};

*/
```

Node* reverseList(Node *head)

{

```
    if(head==NULL || head->next==NULL){

        return head;

    }

    Node *t=reverseList(head->next);

    head->next->next=head;

    head->next=NULL;

    return t;

}

void reorderList(Node* head) {

    // Your code here

     Node *f=head,*s=head;

  while(f && f->next){

      f=f->next->next;

      s=s->next;

  }

  f=s->next;

  s->next=NULL;

  s=head;

     f=reverseList(f);



     Node *t1,*t2;

     Node *h1=s;

     while(f && s)

     {

         t1=s;
         s=s->next;

         t1->next=f;

         t2=f;

         f=f->next;

         t2->next=s;

     }

     //return h1;

}
```