**Merge Sort on Doubly Linked List**

Given Pointer/Reference to the head of a doubly linked list of N nodes, the task is to **Sort the given doubly linked list using Merge Sort**in both **non-decreasing** and **non-increasing** order.

**Example 1:**

**Input:**

N = 8

value[] = {7,3,5,2,6,4,1,8}

**Output:**

1 2 3 4 5 6 7 8

8 7 6 5 4 3 2 1

**Explanation:** After sorting the given

linked list in both ways, resultant

matrix will be as given in the first

two line of output, where first line

is the output for non-decreasing

order and next line is for non-

increasing order.

**Example 2:**

**Input:**

N = 5

value[] = {9,15,0,-1,0}

**Output:**

-1 0 0 9 15

15 9 0 0 -1

**Explanation:** After sorting the given

linked list in both ways, the

resultant list will be -1 0 0 9 15

in non-decreasing order and

15 9 0 0 -1 in non-increasing order.

//{ Driver Code Starts

import java.util.\*;

import java.util.\*;

import java.io.\*;

class Node

{

int data;

Node next, prev;

Node(int key)

{

data = key;

next = prev = null;

}

}

class Driverclass

{

public static void main (String[] args) {

Scanner sc = new Scanner(System.in);

int t = sc.nextInt();

while(t-- > 0)

{

int n = sc.nextInt();

int a1 = sc.nextInt();

Node head = new Node(a1);

Node temp = head;

for (int i = 1;i < n;i++)

{

int a = sc.nextInt();

Node n1 = new Node(a);

n1.prev = temp;

temp.next = n1;

temp = n1;

}

head = new LinkedList().sortDoubly(head);

printList(head);

}

}

public static void printList(Node node)

{

Node temp = node;

while(node != null)

{

System.out.print(node.data + " ");

temp = node;

node = node.next;

}

System.out.println();

while(temp != null)

{

System.out.print(temp.data + " ");

temp = temp.prev;

}

System.out.println();

}

}

// } Driver Code Ends

class LinkedList

{

//Function to sort the given doubly linked list using Merge Sort.

static Node sortDoubly(Node head)

{

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

return head;

}

Node slow = head, fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

Node mid = slow;

slow.prev.next = null;

slow.prev = null;

Node n1 = sortDoubly(head);

Node n2 = sortDoubly(mid);

return merge(n1, n2);

}

static Node merge(Node n1,Node n2){

if(n1 == null)

return n2;

if(n2 == null)

return n1;

Node curr=new Node(0);

if(n1.data<=n2.data){

curr.next=new Node(n1.data);

n1=n1.next;

}

else{

curr.next=new Node(n2.data);

n2=n2.next;

}

Node temp =curr.next;

curr=curr.next;

while (n1 != null && n2 != null) {

if (n1.data <= n2.data) {

Node t = new Node(n1.data);

t.prev = curr;

curr.next = t;

curr = curr.next;

n1 = n1.next;

} else {

Node t = new Node(n2.data);

t.prev = curr;

curr.next = t;

curr = curr.next;

n2 = n2.next;

}

}

if (n1 != null) {

curr.next = n1;

n1.prev=curr;

}

if (n2 != null) {

curr.next = n2;

n2.prev=curr;

}

return temp;

}

}