**Monk And Some Queries**

Monk was asked to answer some queries in an interview. He is given an empty array **A**. Queries are of 4 types:-  
1. **1 X** - Add number **X** to the array **A**.  
2. **2 X** - Remove a single instance of number **X** from the array **A**. If not possible, print "**-1**" without the quotes.  
3. **3** - Find the maximum element in the array **A**.  
4. **4** - Find the minimum element in the array **A**.

**Input:**  
The first line contains the integer **Q**.  
The next **Q** lines will each contain a query like the ones mentioned above.

**Output:**   
For queries **3** and **4**, print the answer in a new line. If the array is empty for query **3** and **4**, then print "**-1**" without the quotes.

**Constraints:**  
1 <= **Q** <= 100000  
1 <= **X** <= 100000

**SAMPLE INPUT**

5

1 5

1 9

1 6

3

2 1

**SAMPLE OUTPUT**

9

-1

**Explanation**

There are 5 queries.  
Query 1 - 5 is added to the array.  
Query 2 - 9 is added to the array.  
Query 3 - 6 is added to the array.  
Query 4 - The maximum element in the array is 9.  
Query 5 - Since there is no element in the array with value 1, so the output is -1.

Solution :

import java.io.BufferedReader;

import java.io.InputStreamReader;

class LL {

Node front = null;

Node rear = null;

class Node {

int data;

Node next;

Node(int d) //allocate memory to node

{

data = d;

next = null;

}

}

public void insert\_prio(int d)

{

Node new\_node = new Node(d);

if(front == null && rear == null)

{

front = new\_node;

rear = new\_node;

}

else {

Node curr = front;

Node last = null;

while(new\_node.data<curr.data && curr.next!=null)

{

last = curr;

curr = curr.next;

}

if(new\_node.data>=curr.data && curr==front) //if curr=front

{

new\_node.next = curr;

//curr.pre = new\_node;

front = new\_node;

}

else if(new\_node.data>=curr.data) // if curr is at middle

{

//Node tmp = curr.pre;

new\_node.next = curr;

//curr.pre = new\_node;

//new\_node.pre = tmp;

last.next = new\_node;

}

else // if curr=rear

{

curr.next = new\_node;

//new\_node.pre = curr;

new\_node.next = null;

rear = new\_node;

}

}

}

public void delete\_ele(int d)

{

Node curr = front;

Node last = null;

while(curr!=null && d<curr.data)

{

last = curr;

curr = curr.next;

}

if(curr==null) // element not present

{

System.out.println("-1");

}

else if(d==curr.data && front==rear) // only 1 element is present

{

front = null;

rear = null;

}

else if(d==curr.data && curr==front) // if curr=front

{

Node tmp\_L = front;

front = front.next;

tmp\_L.next = null;

//front.pre = null;

}

else if(d==curr.data && curr==rear) // if curr=rear

{

rear = last;

//rear = rear.pre;

//tmp\_L.pre = null;

rear.next = null;

}

else if(d==curr.data && curr!=front && curr!=rear) // if d is at middle

{

//Node tmp\_L = curr.pre;

last.next = curr.next;

//tmp\_L.next = tmp\_R;

//tmp\_R.pre = tmp\_L;

curr.next = null;

}

else { // element not present

System.out.println("-1");

}

}

}

class TestClass {

public static void main(String args[] ) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

LL l1 = new LL();

String n1 = br.readLine();

int n = Integer.parseInt(n1);

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

{

String s1[] = br.readLine().split(" ");

if(s1[0].equals("1"))

{

l1.insert\_prio(Integer.parseInt(s1[1]));

}

if(s1[0].equals("2"))

{

l1.delete\_ele(Integer.parseInt(s1[1]));

}

if(s1[0].equals("3"))

{

if(l1.front==null) { System.out.println("-1"); }

else { System.out.println(l1.front.data); }

}

if(s1[0].equals("4"))

{

if(l1.rear==null) { System.out.println("-1"); }

else { System.out.println(l1.rear.data); }

}

}

}

}