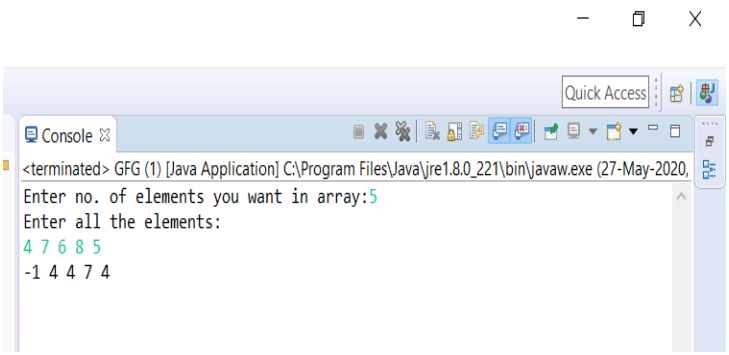
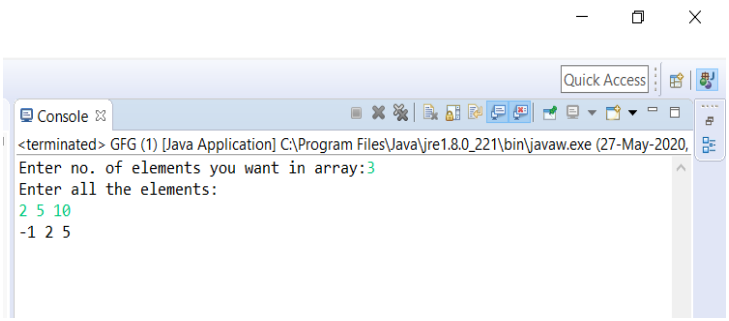
Given an array arr[] of the positive integers of size N, the task is to find the largest element on the left side of each index which is smaller than the element present at that index. Note: If no such element is found then print -1.

```
import ava.util.*;

class GFG{

//Function to find the
//Largest element before
//every element of an array
static void findMaximumBefore(int arr[], int n){

	// Loop to iterate over every
	// element of the array
	for (int i = 0; i<n; i++) {

		int currAns = -1;

		// Loop to find the maximum smallest
		// number before the element arr[i]
		for (int j = i - 1; j>= 0; j--) {
			if (arr[j] >currAns&&
				arr[j] <arr[i]) {
				currAns = arr[j];
			}
		}
		System.out.print(currAns+" ");
	}
}
public static void main(String[] args)
{
	int n;
	Scanner s = new Scanner(System.in);
	 System.out.print("Enter no. of elements you want in array:");
		n = s.nextInt();
	  int arr[] = newint[n];
	   System.out.println("Enter all the elements:");
for(int i = 0; i<n; i++)
    {
        arr[i] = s.nextInt();
    }
	int n1 = arr.length;

	// Function Call
	findMaximumBefore(arr, n1);
}
}
```

OUTPUT:


Quick Access
Console
<terminated> GFG (1) [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (27-May-2020,
Enter no. of elements you want in array:5
Enter all the elements:
4 7 6 8 5
-1 4 4 7 4


Quick Access
Console
<terminated> GFG (1) [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (27-May-2020,
Enter no. of elements you want in array:3
Enter all the elements:
2 5 10
-1 2 5

Write a Java program to implement Binary Tree using the Linked List

```
import java.util.LinkedList;
import java.util.Queue;
public class BinaryTree {
//Represent a node of binary tree
public static class Node{
int data;
Node left;
Node right;

public Node(intdata){
//Assign data to the new node, set left and right children to null
this.data = data;
this.left = null;
this.right = null;
}
}

//Represent the root of binary tree
public Node root;

public BinaryTree(){
root = null;
}

//insertNode() will add new node to the binary tree
public void insertNode(int data) {
//Create a new node
Node newNode = new Node(data);

//Check whether tree is empty
if(root == null){
root = newNode;
return;
}
else {
Queue<Node>queue = new LinkedList<Node>();
//Add root to the queue
queue.add(root);

while(true) {

Node node = queue.remove();
//If node has both left and right child, add both the child to queue
if(node.left != null&&node.right != null) {
queue.add(node.left);
queue.add(node.right);
}
else {
//If node has no left child, make newNode as left child
if(node.left == null) {
node.left = newNode;
queue.add(node.left);
```

```
}
//If node has left child but no right child, make newNode as right child
else {
node.right = newNode;
queue.add(node.right);
}
break;
}
}
}
}

//inorder() will perform inorder traversal on binary search tree
public void inorderTraversal(Node node) {

//Check whether tree is empty
if(root == null){
System.out.println("Tree is empty");
return;
}
else {

if(node.left!= null)
inorderTraversal(node.left);
System.out.print(node.data + " ");
if(node.right!= null)
inorderTraversal(node.right);

}
}

public static void main(String[] args) {

BinaryTreebt = newBinaryTree();
//Add nodes to the binary tree

bt.insertNode(1);
//1 will become root node of the tree
System.out.println("Binary tree after insertion");
//Binary after inserting nodes
bt.inorderTraversal(bt.root);

bt.insertNode(2);
bt.insertNode(3);
//2 will become left child and 3 will become right child of root node 1
System.out.println("\nBinary tree after insertion");
//Binary after inserting nodes
bt.inorderTraversal(bt.root);

bt.insertNode(4);
bt.insertNode(5);
//4 will become left child and 5 will become right child of node 2
System.out.println("\nBinary tree after insertion");
//Binary after inserting nodes
bt.inorderTraversal(bt.root);
```

```
        bt.insertNode(6);
        bt.insertNode(7);
        //6 will become left child and 7 will become right child of node 3
        System.out.println("\nBinary tree after insertion");
        //Binary after inserting nodes
        bt.inorderTraversal(bt.root);

        }
}
```

OUTPUT: