DSA23 Assignment 2

(Stacks, Queues, Binary Trees and Binary Search trees)

Deadline: April 13, 2023

1 Nest Checker and Palindrome using stacks(50 points + 50 points)

For this question you have to perform two operations, first is to check if a string is balanced or not. A string is called a balanced string when:

- Iff it has brackets ({, [, (,),], }), the opening and closing brackets are in a correct nested form. For example, '[{()}]' is a balanced string, while '[{(][)]}]' is not.

The second is to check if its a palindrome. You can find the definition of palindrome here .

1.1 Input Format

Input will consist of multiple lines.

- The first line will contain the number of test cases **T**.
- The subsequent lines will contain the strings. There are no spaces inside the strings, therefore you can use normal *scanf()* to read the input.

1.2 Output Format

For each test case

- Print **"Balanced"** if the string is only balanced.
- Print **"Palindromic"** if the string is only Palindromic.
- Print **"Balanced and Palindromic"** if it is both.
- Print **"-1"** if it is none.

Example 1

Input

2
[{abcde}]
{(abcba)}

Output

Balanced
Balanced

Example 2

Input

3
[{abab}]
{(absdfsdcba)]}
abcdcba

Output

Balanced
-1
Palindromic

1.3 Explanation

The first example has two strings as input. The first string has brackets in an nested order and is not a palindrome, hence the corresponding output is **"Balanced"**. Similarly for the second input the string has brackets in a correct order as well as it is not a palindrome, hence the output is **"Balanced"**

For the second example, the first input is balanced, the second input is not balanced nor Palindromic hence the corresponding output is **"-1"**. The third input doesn't have any brackets and is palindromic, hence the output is **"Palindromic"**.

1.4 Constraints

$0 \leq T \leq 100$
$0 \leq L \leq 10000$ where L is length of input string.

2 Circular Deque (100 Points)

In this question, you have to implement an ADT called Circular Deque using queue data structure . A deque is known as double-ended queue. You can check more about Deque here. Now You have to implement a Cicrular Deque which performs the required Operations.

2.1 Functions To Implement:

- `void Push(Queue head,int val)`: Creates a new node containing the integer val and inserts at the end of the circular Deque.
- `int Pop(Queue head)`: Remove the front element in the Circular Deque and return it. If there is no element in Circular Deque, return -1.
- `void Inject(Queue head,int val)`: Creates a new node containing the integer val and adds it to the front end of the circular Deque.
- `int popRear(Queue head)`: Remove the last element in the Circular Deque and return it.If there is no element in Circular Deque, return -1.
- `void Print(Queue head)`: Prints the Circular Deque from the first element. If deque is empty, Print -1.
- `void PrintReverse(Queue head)`: Prints the Circular Deque in reverse manner. If deque is empty, Print -1.
- `int findElem(Queue head,int pos)`: Find the element in the given position and return it. If there is no element present in given position, return -1. Consider the Circular Deque as 1-indexed.
- `void removeKElems(Queue head,int k)`: Remove the front k elements in the Circular Deque. If k is greater than the size of deque, remove all the elements.

Note: 1. After completion of each operation the Circular Deque must necessarily be circular.

2.2 Input and Output

The first line contains T, the number of operations that need to be performed.
$1 \leq T \leq 10000$.
The next line consists of a string indicating the OPERATION to be executed.
The next few lines give the data needed for the operation i.e.

Operation name	**Corresponding function**
OPER1	*Push*
OPER2	*Pop*
OPER3	*Inject*
OPER4	*popRear*
OPER5	*Print*
OPER6	*PrintReverse*
OPER7	*findElem*
OPER8	*removeKElems*

OPERATION	**INPUT FORMAT**	**CONSTRAINTS**	**OUTPUT FORMAT**
OPER1	First line contains an integer n that must be inserted into end of the circular Deque.	$1 \leq n \leq 10^6$	
OPER2			Print the popped element from circular Deque .
OPER3	First line contains an integer n that must be inserted into front of the circular Deque.	$1 \leq n \leq 10^6$	
OPER4			Print the Rear popped element from circular Deque .
OPER5			Print all the elements in the circular Deque in order.
OPER6			Print all the elements in the circular Deque in Reverse order.
OPER7	First line contains an integer n. This is the position of an element.	$1 \leq n \leq 10^4$	Print the element in the nth position.
OPER8	First line contains an integer k. Where k is the number of elements to be remove.	$1 \leq k < 10^6$	

2.3 Example Test Cases

Input	Output
15	
OPER1 4	
OPER3 5	
OPER1 2	
OPER1 6	
OPER2	5
OPER4	6
OPER1 1	
OPER5	4 2 1
OPER3 7	
OPER6	1 2 4 7
OPER3 3	
OPER7 4	2
OPER8 3	
OPER5	2 1
OPER7 3	-1

2.4 Explanation

We have:

1. Push 4 : Inserts 4 at end. The current state of the circular Deque will be:

   $$4$$

2. **Inject 5** : Inserts 5 at front. The current state of the circular lDeque will be:

   $$5 - 4$$

3. **Push 2** : Inserts 2 at end. The current state of the circular Deque will be:
$$5 - 4 - 2$$

4. **Push 6** : Inserts 6 at end. The current state of the circular Deque will be:
$$5 - 4 - 2 - 6$$

5. **Pop** : Remove the front element and return it. The current state of the circular Deque will be:
$$4 - 2 - 6$$

6. **popRear** : Remove the rear element and return it. The current state of the circular Deque will be:
$$4 - 2$$

7. **Push 1** : Inserts 1 at end. The current state of the circular Deque will be:
$$4 - 2 - 1$$

8. **Print** : Print the current circular deque.

9. **Inject 7** : Inserts 7 at front. The current state of the circular Deque will be:
$$7 - 4 - 2 - 1$$

10. **PrintReverse** : Prints the circular deque in reverse manner.

11. **Inject 3** : Inserts 3 at front. The current state of the circular Deque will be:
$$3 - 7 - 4 - 2 - 1$$

12. **findElem 4** : Print the element at Position 4 which is 2.

13. **removeKElems 3** : Remove the front 3 elements. The current state of the circular Deque will be:
$$2 - 1$$

14. **Print** : Print the current circular deque.

15. **findElem 3** : The given position is greater than the size of Circular Deque. Hence return -1.

3 Trees - The Story of Binary Trees(100 points)

The objective of this question is to practice traversals in trees.

There are 2 easy, basic subproblems to be solved, which will get you familiar with the basic strategies to tackle most tree questions moving forward

3.1 Traversal in Trees(50 points)

1. **Problem Explanation**
   Trees can be traversed in several different ways, as they are non-linear data structures. Of these; one way is "Beautiful Traversal"
   Given a Tree like this:

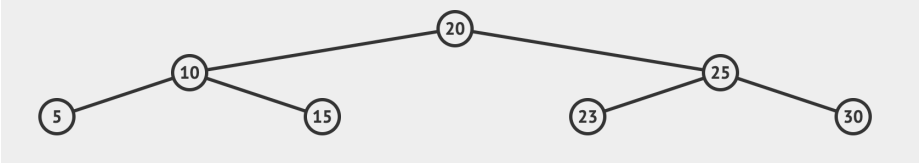

Figure 1: Example Tree for 3.1.1

   The "Beautiful" traversal is: 20 25 10 5 15 23 30

2. **Input and Output**

   Input

   - The First Line consists of a number of test cases - an integer variable - **T**
   - The Second Line consists of **N** - the number of Nodes in the tree
   - The Third Line consists of **N** space separated integer numbers, The Elements of the tree, n_i.

   Output

   - The First Line of the output contains N space-separated integer numbers which represent the "Beautiful"-traversal(Zig-zag Level order Traversal) of the tree

Constraints

- $1 \leq T \leq 10$
- $1 \leq N \leq 10^5$
- $1 \leq n_i \leq 10^9$

3. **Example TestCase**

Input	output
1	20 25 10 5 15 23 30
7	
20 10 25 5 15 23 30	

3.2 Traversal to Trees(50 points)

1. **Problem Explanation**
   Construct Tree from given Inorder and Preorder traversals
   Hint - In a Preorder sequence, the leftmost element is the root of the tree. By searching for root in the Inorder sequence, we can find out all elements on the left side of it, is in the left subtree and elements on right in the right subtree.

   Say, Inorder sequence: D B E A F C ;Preorder sequence: A B D E C F Tree =

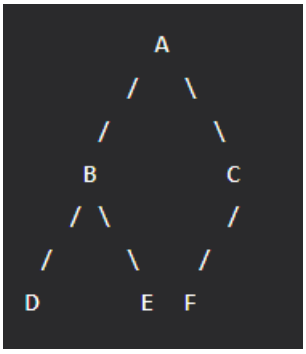

Figure 2: Output Example Tree for 3.1.3

2. **Input and Output**

Input

- The First Line consists of a number of test cases - an integer variable - **T**

- The Second Line consists of **N** - the number of Nodes in the tree
- The Third Line consists of **N** space separated integer numbers, The Elements of the tree in the Inorder Sequence, n_i
- The Fourth Line consists of **N** space-separated integer numbers, The Elements of the tree in the Preorder Sequence. m_i

Output

- The First Line of the output contains N space-separated integer numbers that represent the Tree.

Constraints

- $1 \leq T \leq 10^1$
- $1 \leq N \leq 10^3$
- $1 \leq n_i \leq 10^9$
- $1 \leq m_i \leq 10^9$

3. **Example TestCase**

Input	**output**
1	1 2 3 4 5 6
6	
4 2 5 1 6 3	
1 2 4 5 3 6	

4 King and the Crisis (100 Points)

Once upon a time, there was a kingdom ruled by a wise King who loved nature and often went on forest walks. During one of his walks, he came across a beautiful **Tree** with several branches and leaves. The king, being a nature lover, decided to keep the tree as it was and even gave orders to his workers to take care of it.

Years passed by, and the tree grew taller, stronger, and even bore fruits. However, the kingdom was facing a financial crisis, and the king decided to take the tree's fruits and sell them so as a measure to overcome the crisis. But the workers, who had grown attached to the tree, requested the king to spare it.

The king thought for a while and came up with an idea. He decided to make use of the tree's branches to create a **Binary Search Tree** with every fruit hanging from those branches (each Node of the BST) having different **Values V**, hoping to use it as a source of income for the kingdom.

However, the king soon realized that the tree was not ordinary, and he didn't want to harm it in any way. He asked to workers to modify the tree such that each fruit's price (Node's Value) gets updated to the **sum of all the values smaller or equal to the current fruit**. The workers spent days trying to figure out a way to fulfil their Kings order but couldn't find a way out.

You are a good friend of one of the workers got a call to assist. Help the workers to find the **Maximum Sale** possible out of this tree. Maximum sale is the sum of all values in the updated Binary Search Tree.

4.1 Input and Output

Input

First line contains integer number $N(1 \leq N \leq 10^6)$ - number of fruits (Nodes) in the tree.

Second line contains N distinct integers, $v_1, v_2 ... v_n(-10^6 \leq v_i \leq 10^6)$ each denoting value of Binary Search Tree in a top - to - bottom (root - to - leaf) manner.

Output

Each Output contains 2 lines -

- Line 1: Updated Binary Search Tree in a top - to - bottom manner.
- Line 2: Max Possible Sale (Sum of all the values of Updated Binary Search Tree)

4.2 Example 1

Input:
7
20 10 25 5 15 23 30

Output:
50 15 98 5 30 73 128
399

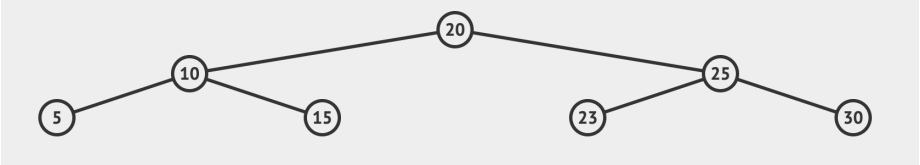

Figure 3: Input Binary Search Tree

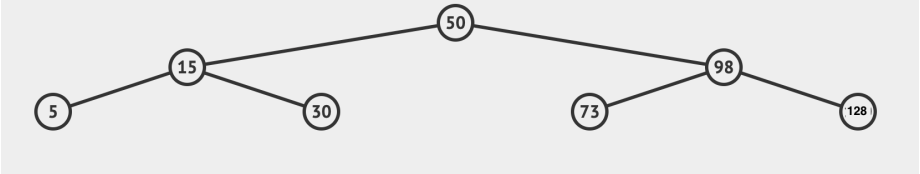

Figure 4: Updated Binary Search Tree

4.2.1 Explanation for BST Update

1. **For Node 20** : Updated Value will be some of all values smaller or equal than 20.
$$5 + 10 + 15 + 20 => 50$$
2. **For Node 10** : Updated Value will be some of all values smaller or equal than 10.
$$5 + 10 => 15$$
3. **For Node 25** : Updated Value will be some of all values smaller or equal than 25.
$$5 + 10 + 15 + 20 + 23 + 25 => 98$$
4. **For Node 5** : Updated Value will be some of all values smaller or equal than 5.
$$5 => 5$$

5. **For Node 15** : Updated Value will be some of all values smaller or equal than 15.
$$5 + 10 + 15 => 30$$

6. **For Node 23** : Updated Value will be some of all values smaller or equal than 23.
$$5 + 10 + 15 + 20 + 23 => 73$$

7. **For Node 30** : Updated Value will be some of all values smaller or equal than 30.
$$5 + 10 + 15 + 20 + 23 + 25 + 30 => 128$$

Sum of Updated BST :

$$50 + 15 + 98 + 5 + 30 + 73 + 128 => 399$$

4.3 Example 2

Input:
11
8 3 10 1 6 14 4 7 13 11 12

Output:
29 4 39 1 14 89 8 21 75 50 62
392

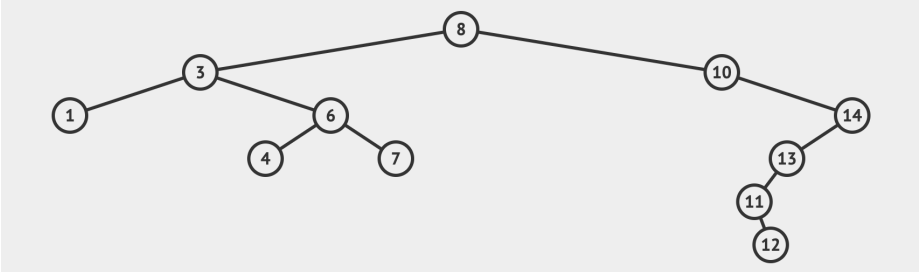

Figure 5: Input Binary Search Tree

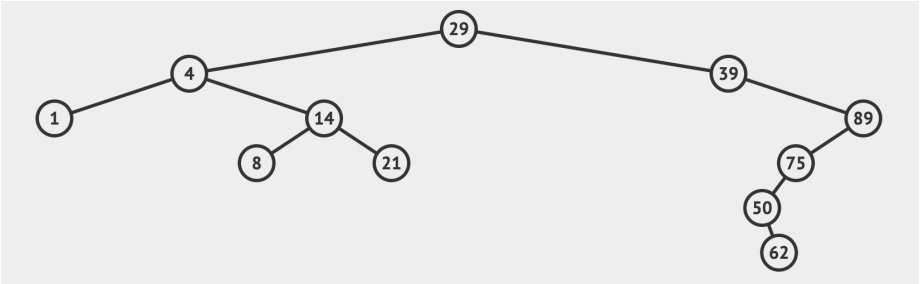

Figure 6: Updated Binary Search Tree

4.4 Functions To Implement:

- **Node* ListToBST (int *arr)**: Creates a new Binary Search Tree containing the integer values in the List given as input and returns Node pointer to the root node of the tree.
- **void ModifyBST (Node* BST)**: Takes root node of Input BST as argument and Modifies in **inplace** the same as per the question.

  NOTE: You need not need to follow the exact function definition mentioned above. It's just get you a hint of the structure. But this need to be coded using BST, any other approach that doesn't involve BST will be straight given **0**.