**3. Eve and his System**

Given a linked list where every node represents a linked list and contains two pointers of its type:

* Pointer to next node in the main list (we call it ‘right’ pointer in below code)
* Pointer to a linked list where this node is head (we call it ‘down’ pointer in below code).
* All linked lists are sorted.
* Write a function flatten() to flatten the lists into a single linked list. The flattened linked list should also be sorted.

**Input Format**

First line contains an integer N denoting the number of head nodes connected to each other.  
Second line contains N space separated numbers (M1, M2…Mn) denoting the number of elements in linked lists starting with each head.  
Third line contains all the elements of the linked list starting with 1st head node and covering all the elements through its down pointer, then 2nd head node and covering all its elements through down pointer and so on till the last head node of the linked list.

Constraints

0<=N<=50 1<=Mi<=20 1<=Element of linked list<=1000

**Output Format**

Changes must be done in the Original LinkedList so no necessary to return head of the LinkedList.

Simply display the new LinkedList in a space separated manner.

**Sample Input**

**4**

**4 2 3 4**

**5 7 8 30 10 20 19 22 50 28 35 40 45**

**Sample Output**

**5 7 8 10 19 20 22 28 30 35 40 45 50**

**Explanation**

**5 -> 10 -> 19 -> 28**

**| | | |**

**V V V V**

**7 20 22 35**

**| | |**

**V V V**

**8 50 40**

**| |**

**V V**

**30 45**

**Test case 2:**

**7**

**3 15 3 12 15 13 10**

**11 25 49 26 26 28 39 39 42 42 52 53 53 74 80 81 86 94 32 36 73 8 25 32 51 56 65 68 81 83 87 91 99 4 7 7 16 19 35 42 52 58 68 73 80 84 86 98 1 3 6 11 19 20 31 54 59 63 68 84 95 9 10 19 22 59 68 81 90 92 92**

**Output:**

**1 3 4 6 7 7 8 9 10 11 11 16 19 19 19 20 22 25 25 26 26 28 31 32 32 35 36 39 39 42 42 42 49 51 52 52 53 53 54 56 58 59 59 63 65 68 68 68 68 73 73 74 80 80 81 81 81 83 84 84 86 86 87 90 91 92 92 94 95 98 99**

**Test case 3:**

**16**

**13 2 7 4 9 1 9 11 8 5 12 15 18 17 14 16**

**6 7 18 24 36 42 43 50 68 83 83 88 98 15 18 29 33 37 41 79 83 91 14 60 63 84 8 12 42 42 46 61 87 90 97 12 13 27 36 54 83 94 96 98 98 8 18 34 39 66 68 84 85 90 91 98 7 17 34 47 53 62 95 96 10 11 41 87 94 8 12 13 29 38 38 44 45 49 59 61 72 3 4 8 11 14 30 42 43 52 56 60 65 76 78 88 13 19 29 35 36 38 41 48 57 66 69 72 72 72 76 81 89 91 3 9 12 21 26 33 35 44 53 58 61 72 85 87 90 94 95 2 19 23 37 47 49 49 60 61 68 77 81 81 84 3 17 20 37 38 43 47 55 65 70 70 83 83 89 95 97**

**Output:**

**2 3 3 3 4 6 7 7 8 8 8 8 9 10 11 11 12 12 12 12 13 13 13 14 14 15 17 17 18 18 18 19 19 20 21 23 24 26 27 29 29 29 30 33 33 34 34 35 35 36 36 36 37 37 37 38 38 38 38 39 41 41 41 42 42 42 42 43 43 43 44 44 45 46 47 47 47 48 49 49 49 50 52 53 53 54 55 56 57 58 59 60 60 60 61 61 61 61 62 63 65 65 66 66 68 68 68 69 70 70 72 72 72 72 72 76 76 77 78 79 81 81 81 83 83 83 83 83 83 84 84 84 85 85 87 87 87 88 88 89 89 90 90 90 91 91 91 94 94 94 95 95 95 96 96 97 97 98 98 98 98**

**4 Alice and Merg Game**

You are given two binary search trees (BSTs), T1 and T2, which may contain overlapping nodes. Write a code in your preferred programming language to merge these two BSTs into a single BST. The merged BST should include all the nodes from T1 and T2 with their values in sorted order.

**Input:**

First line contains the values of all the nodes in the binary tree **T1** in pre-order format where true suggest the node exists and false suggests it is NULL

Second line contains the values of all the nodes in the binary tree **T2** in pre-order format where true suggest the node exists and false suggests it is NULL

**Output Format:**

Print the merged BST.

**Input Testcase 1**

8 true 5 true 2 false false true 7 false false

10 true 15 true 12 false false true 18 false false

**Output:**

10 true 8 true 5 true 2 false false true 7 false false false true 15 true 12 false false true 18 false false

**Input Testcase 2**

10 true 5 true 3 false false true 7 false false

15 true 12 true 11 false false true 18 false false

**Output**:

15 true 10 true 5 true 3 false false true 7 false false true 12 true 11 false false false true 18 false false

**Explanation:**

In the merged BST, all nodes from both trees are included and sorted. Note that the value 4 is chosen as the root since it's the median value between the minimum value in T2 and the maximum value in T1.

**Constraints**:

Each tree may contain duplicate values.

Assume that there are no duplicate values between T1 and T2.

You should not modify the structure of either tree; instead, construct a new tree.

**Hints:**

Consider using an iterative approach to merge the trees while maintaining the BST properties.

Utilize in-order traversal to collect nodes in sorted order and construct the merged BST.

This problem challenges you to not only understand BST properties but also to merge two tree structures while ensuring the resulting tree maintains the BST property and is efficiently constructed.

**Extra Testcases**

Input

8 true 5 true 2 false false true 7 false false

10 true 15 true 12 false false true 18 false false

Output:

10 true 8 true 5 true 2 false false true 7 false false false true 15 true 12 false false true 18 false false

Input

10 true 5 true 3 false false true 7 false false

15 true 12 true 11 false false true 18 false false

Output:

15 true 10 true 5 true 3 false false true 7 false false true 12 true 11 false false false true 18 false false

Input:

7 true 3 true 2 false false true 5 false false

8 true 6 true 4 false false true 9 false false

Output:

8 true 7 true 6 true 3 true 2 false false true 5 false false true 4 false false true 9 false false

Input:

5 true 3 true 2 false false true 4 false false

6 true 7 true 8 false false true 9 false false

Output:

6 true 5 true 3 true 2 false false true 4 false false true 7 true 8 false false true 9 false false

Input:

T1 = 10 true 5 true 2 false false true 7 false false

T2 = 10 true 5 true 2 false false true 7 false false

Output:

10 true 10 true 5 true 5 true 2 false false true 2 false false true 7 false false true 7 false false

**5. Diana's Recovery**

Diana accidentally swapped two nodes in a binary search tree (BST) while working on a project. She needs to recover the tree without changing its structure by swapping the nodes back to their correct positions.

You are given the root of a binary search tree (BST), where the values of exactly two nodes of the tree were swapped by mistake. Recover the tree without changing its structure.

**Input Format:**

First line contains the values of all the nodes in the binary tree in pre-order format where true suggest the node exists and false suggests it is NULL

**Output Format:**

Print the new tree.

**Sample Test Case 1:**

Input:

8 true 6 true 5 false false true 7 false false true 10 false true 12 false false

Output:

8 true 6 true 5 false false true 7 false false true 10 true 12 false falsefalse

Explanation: The binary tree is:

8

/ \

6 10

/ \ \

5 7 12

**Sample Test Case 2:**

Input:

12 true 8 true 5 false false true 10 true 9 false false true 11 false false true 15 false true 14 false false true 17 false false

Output:

12 true 8 true 5 false false true 10 true 9 false false true 11 false false true 15 true 14 false false true 17 false falsefalse

Explanation: The binary tree is:

12

/ \

8 15

/ \ / \

5 10 14 17

/ \

9 11

**Constraints:**

The number of nodes in the tree is in the range `[2, 1000]`.

2^31 <= Node.val<= 2^31 - 1

**Extra Test Cases:**

Input:

8 true 6 true 5 false false true 7 false false true 10 false true 12 false false

Output:

8 true 6 true 5 false false true 7 false false true 10 true 12 false falsefalse

Input:

12 true 8 true 5 false false true 10 true 9 false false true 11 false false true 15 false true 14 false false true 17 false false

Output:

12 true 8 true 5 false false true 10 true 9 false false true 11 false false true 15 true 14 false false true 17 false falsefalse

Input:

25 true 20 true 15 false false true 22 false false true 30 true 28 false false true 35 false false

Output:

25 true 20 true 15 false false true 22 false false true 30 true 35 false false true 28 false false

Input:

18 true 15 true 10 true 5 false false true 12 false false true 17 true 16 false false true 20 true 19 false false true 25 false false

Output:

18 true 15 true 10 true 5 false false true 12 false false true 20 true 19 false false true 17 true 16 false false true 25 false false

Input :

50 true 40 true 35 false false true 45 false false true 60 true 55 false false true 65 false false

Output: 50 true 40 true 35 false false true 45 false false true 60 true 55 false false true 65 false false