**EASY**

**1 Sarah and his Gallary**

Sarah is organizing her photo gallery app. She wants to view her photos in reverse order, so she needs to reverse the linked list that represents the sequence of photos.

Given a singly linked list, reverse the list and return the new head.

**Input Format**

First line Take Input n

Next Line Contains N elements of LinkedList.

**Output Format**

Print LinkedList in reverse order.

**Sample Test Case 1:**

Input:

5

1 2 3 4 5

Output:

5 4 3 2 1

Explanation: The original list is 1 -> 2 -> 3 -> 4 -> 5. Reversing it, the last node (5) becomes the first node, followed by 4, 3, 2, and 1.

**Sample Test Case 2:**

Input:

3

0 0 0

Output

0 0 0

Explanation: The original list is 0 -> 0 -> 0 ->NULL. Reversing it, the last node (0) becomes the first node, followed by 0, and 0.

**Constraints**:

The number of nodes in the list is in the range `[0, 5000]`.

1000 <= Node.Val<= 1000

**Extra Test Cases:**

Input:

1

1

Output:

1

Input:

2

5 7

Output:

7 5

Input:

4

10 20 5 7

Output:

7 5 20 10

Input:

6

1 2 3 4 5 6

Output:

6 5 4 3 2 1

Input:

3

-9 11 4

Output:

4 11 -9

**2 John And Game**

John is building a game where characters move in a loop. To avoid infinite loops, he needs to detect if the linked list representing the movement path has a cycle.

Given a linked list, determine if it has a cycle in it.

**Input Format**

First line contains n elements of LinkedList.

**Output Format**

Print true or false

**Sample Test Case 1:**

Input: 3 -> 2 -> 0 -> -4 (tail connects to node index 1)

Output: true

**Explanation**: The list 3 -> 2 -> 0 -> -4 has a cycle because the last node (-4) connects back to the node at index 1 (node with value 2).

**Sample Test Case 2:**

Input: 1 -> 1 (cycle back to itself)

Output: true

**Explanation**: The list 1 -> 1 has a cycle because the cycle back to itself.

**Constraints:**

The number of nodes in the list is in the range [0, 10000].

100000 <= Node.val<= 100000

**Extra Test Cases:**

Input: 1 -> NULL

Output: false

Input: 1 -> 2 -> 3 -> 4 -> 5 -> NULL

Output: false

Input: 1 -> 2 -> 3 -> 4 -> 2 (cycle back to node 2)

Output: true

Input: 1 -> 1 (cycle back to itself)

Output: true

Input: NULL

Output: false

**3 Emily and Merge**

Emily is combining two sorted photo albums into one. She needs to merge two sorted linked lists to maintain the order.

Given two sorted linked lists, merge them into one sorted linked list.

**Input Format**

First line contains n (Number of elements).

Second Line Contains n elements of LinkedListA

Third line contains k(Number of elements).

Fourth line contains k elements of LinkedListB

**Output Format**

Print the new LL after merging the elements.

**Sample Test Case 1:**

Input:

3

1 2 4

3

1 3 4

Output: 1 1 2 3 4 4

**Explanation**: The two sorted lists are 1 2 4 and 1 3 4. Merging them in order gives 1 1 2 3 4 4.

**Sample Test Case 2:**

Input:

3

-1 5 10

3

0 3 7

Output:

-1 0 3 5 7 10

**Explanation**: The two sorted lists are -1 5 10 and 0 3 7. Merging them in order gives -1 0 3 5 7 10.

**Constraints:**

The number of nodes in both lists is in the range [0, 50].

100 <= Node.val <= 100

**Extra Test Cases:**

Input:

1

0

1

0

Output:

0 0

Input:

1

1

1

0

Output:

0 1

Input:

5

1 2 3 4 5

1

0

Output: 0 1 2 3 4 5

Input:

3

1 5 10

3

0 3 7

Output: - 1 0 3 5 7 10

Input:

3

10 -5 0

3

1 2 3

Output: - 10 -5 0 1 2 3

**4 Remove N-th Node from End of List**

Alice needs to remove the N-th most recent message from her chat history, which is stored as a linked list.

Given the head of a linked list, remove the n-th node from the end of the list and return its head.

**Input Format**

First line take input k and N

Next Line contains k elements of LinkedList.

Second line contains value of n.

**Output Format**

Print LL after deleting nth node.

**Sample Test Case 1:**

Input:

5 2

1 2 3 4 5

Output:

1 2 3 5

**Explanation**:

The list is 1 -> 2 -> 3 -> 4 -> 5. The 2nd node from the end is 4. Removing it results in 1 -> 2 -> 3 -> 5.

**Sample Test Case 2:**

Input

5 1

1 2 3 4 5

Output: 1 2 3 4

**Explanation**:

The list is 1 -> 2 -> 3 -> 4 -> 5. The 1st node from the end is 5. Removing it results in 1 -> 2 -> 3 -> 4.

**Constraints:**

The number of nodes in the list is in the range [1, 30].

1 <= Node.val<= 100

1 <= n <= the number of nodes in the list

**Extra Test Cases:**

Input:

5 3

1 2 3 4 5

Output:

1 2 4 5

Input:

5 5

1 2 3 4 5

Output

2 3 4 5

Input:

1 1

Output:

Input:

2 1

1 2

Output:

1

Input:

3 3

1 2 3

Output:

2 3

**5 David his daily activities**

David wants to check if the sequence of his daily activities (stored as a linked list) is the same forwards and backwards, i.e., a palindrome.

Given a singly linked list, determine if it is a palindrome.

**Input Format**

First line contains n (Number of elements)

Second line contains n elements of LinkedList.

**Output Format**

Print true or false

**Sample Test Case 1:**

Input:

4

1 2 2 1

Output: true

**Explanation**: The list 1 2 2 1 is the same forwards and backwards, making it a palindrome.

**Sample Test Case 2:**

Input:

4

1 2 3 1

Output: false

**Explanation**: The list 1 2 3 1 is not the same forwards and backwards, not making it a palindrome

**Constraints:**

1<=n<=100000

-10^9 <=Node.val <= 10^9

**Extra Test Cases:**

Input:

2

1 2

Output: false

Input:

5

1 2 3 2 1

Output: true

Input:

1

1

Output: true

Input:

9

1 2 3 4 5 4 3 2 1

Output: true

Input:

4

1 0 0 1

Output: true

**6 Sophia's Hierarchy**

Sophia wants to visualize her organization's hierarchy. Each level of the hierarchy should be displayed on a new line. She needs to perform a level order traversal of the binary tree representing the hierarchy.

Given a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

**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 tree level order traversal

**Sample Test Case 1:**

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: 8 6 10 5 7 9 12

**Explanation**:

The binary tree is:

8

/ \

6 10

/ \ \

5 7 12

**Sample Test Case 2:**

Input: 8 true 10 true 6 false false true 7 false false true 5 false true 3 false false

Output: 8 10 6 7 5 3

**Explanation**:

The binary tree is:

8

/ \

10 5

/ /

6 3

/

7

**Constraints**:

The number of nodes in the tree is in the range [0, 2000].

1000 <= Node.val<= 1000

**Extra Test Cases:**

Input:

8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output:

8 6 10 5 7 9 12

Input:

8 true 10 true 6 false false true 7 false false true 5 false true 3 false false

Output:

8 10 6 7 5 3

Input:

12 true 8 true 5 true 3 false false true 7 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 8 15 5 10 14 17 3 7 9 11

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 15 20 10 17 25 5 12 16 19

Input:

25 true 20 true 15 true 10 false false true 22 false true 21 false false true 30 true 28 false false true 35 false false

Output:

25 20 30 15 22 28 35 10 21

**7 Liam Database**

Liam is working on a database system that uses a binary search tree (BST) to store data. He needs to verify that the tree structure is correct and adheres to the BST properties.

Given a binary tree, determine if it is a valid binary search tree (BST).

**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 true or false

**Sample Test Case 1:**

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: true

**Explanation**: The binary tree is:

8

/ \

6 10

/ \ \

5 7 12

**Sample Test Case 2:**

Input: 8 true 10 true 6 false false true 7 false false true 5 false true 3 false false

Output: false

**Explanation**: The binary tree is:

8

/ \

10 5

/ /

6 3

/

7

**Constraints:**

The number of nodes in the tree is in the range [1, 10000].

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

**Extra Test Cases:**

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: true

Input: 8 true 10 true 6 false false true 7 false false true 5 false true 3 false false

Output: false

Input: 12 true 8 true 5 true 3 false false true 7 false false true 10 true 9 false false true 11 false false true 15 false true 14 false false true 17 false false

Output: true

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: false

Input: 25 true 20 true 15 true 10 false false true 22 false true 21 false false true 30 true 28 false false true 35 false false

Output: true

**8. Invert Binary Tree**

Rachel is working on a graphics rendering system and needs to invert a binary tree to apply a mirror effect to the image.

Invert a binary tree. Inversion means swapping the left and right child of all nodes in the tree.

**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 preorder traversal of inverted tree.

**Sample Test Case 1:**

Input: 8 true 6 true 5 true 3 false false true 7 false false true 10 false false

Output: 8 true 10 true 6 false false true 7 false false true 5 false true 3 false false

**Explanation**: The binary tree is:

8

/ \

6 10

/ \

5 7

/

3

The inverted binary tree is:

8

/ \

10 6

/ \

7 5

\

3

**Sample Test Case 2:**

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: 8 true 10 true 12 false false true 6 true 7 false false true 5 false true 9 false false

**Explanation**: The binary tree is:

8

/ \

6 10

/ \ \

5 7 12

\

9

The inverted binary tree is:

8

/ \

10 6

/ / \

12 7 5

\

9

**Constraints:**

The number of nodes in the tree is in the range `[0, 100]`.

100 <= Node.val<= 100`

**Extra Test Cases:**

Input: 8 true 6 true 5 true 3 false false true 7 false false true 10 false false

Output: 8 true 10 true 6 false false true 7 false false true 5 false true 3 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 20 true 15 true 25 false false true 19 true 17 false false true 10 true 12 false false true 5 false true 16 false false

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: 8 true 10 true 12 false false true 6 true 7 false false true 5 false true 9 false false

Input: 8 true 6 true 5 false true 7 false false true 10 false true 9 false false true 12 false false

Output: 8 true 10 true 12 false false true 6 true 7 false false true 5 false true 9 false false

**9.Symmetric Tree**

Oliver is working on a symmetrical design for a project and needs to check if a given binary tree is symmetric around its center.

Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

**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 true or false

**Sample Test Case 1:**

Input: 15 true 10 true 7 true 5 false false true 8 false false true 10 false false true 7 true 5 false false true 16 false false

Output: false

Explanation: The binary tree is:

15

/ \

10 10

/ \ / \

7 7 7 7

/ \ / \

5 8 8 16

This tree is non symmetric.

**Sample Test Case 2:**

Input: 15 true 10 true 7 true 5 false false true 8 false false true 10 false false true 7 true 5 false false true 15 false false

Output: true

Explanation: The binary tree is:

15

/ \

10 10

/ \ / \

7 7 7 7

/ \ / \

5 8 8 5

This tree is symmetric.

**Constraints:**

The number of nodes in the tree is in the range `[1, 1000]`.

100 <= Node.val<= 100`

**Extra Test Cases:**

Input: 15 true 10 true 7 true 5 false false true 8 false false true 10 false false true 7 true 5 false false true 16 false false

Output: false

Input: 15 true 10 true 7 true 5 false false true 8 false false true 10 false false true 7 true 5 false false true 15 false false

Output: true

Input: root = 0 false false

Output: true

Input: root = 1 1 false false true 1 false false

Output: true

Input: root = 1 false false

Output: true

**10. Same Tree**

Sam is comparing two versions of a document stored in binary tree structures. He needs to check if the two binary trees are exactly the same.

Given the roots of two binary trees `p` and `q`, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

**Input format:**

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

**Output format:**

Print true and false

**Sample Test Case 1:**

Input:

10 true 5 true 3 true 1 false false true 4 false false true 7 true 6 false false true 8 false false

10 true 5 true 3 true 1 false false true 4 false false true 7 true 6 false false true 8 false false

Output: true

Explanation: The binary trees are:

**Tree p:**

10

/ \

5 7

/ \ /

3 6

/ /

1 4

\

8

**Tree q:**

10

/ \

5 7

/ \ /

3 6

/ /

1 4

\

8

Both trees are the same.

**Sample Test Case 2:**

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output: true

Explanation: The binary trees are:

Tree p:

15

/ \

10 20

/ \ / \

7 12 17 25

/ \ \

5 9 13

/

11

Tree q:

15

/ \

10 20

/ \ / \

7 12 17 25

/ \ \

5 9 13

/

11

Both trees are the same.

**Constraints:**

The number of nodes in both trees is in the range `[0, 100]`.

10^4 <= Node.val<= 10^4`

**Test Cases:**

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output: true

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output:

true

Input

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

15 true 10 true 7 true 5 false false 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output: false

Input:

7 true 6 true 5 false 5 false false true 5 true 5 false false true 6 false false

7 true 6 true 5 false false true 6 true 5 false false true 5 true 5 false false true 6 false false

Output: false

Input:

7 true 6 true 5 false false true 6 true 5 false false true 5 false false true 6 true 5 false false

7 true 6 true 5 false false true 6 true 5 false falsefalse true 6 true 5 false false

Output: false

**Medium**

**1 Calculate their sum**

**1 Calculate their sum**

In a financial application, numbers are represented as linked lists where each node contains a single digit. These numbers are stored in reverse order. Calculate their sum.

Given two non-empty linked lists representing two non-negative integers, add the two numbers and return it as a linked list.

**Input Format**

First line contains input n

second line contains n elements of LinkedListA.

Third line contains input k

next line contains k elements of LinkedListB

**Output Format**

Print the new LL after adding the elements.

**Sample Test Case 1:**

Input:

3

2 4 3

3

5 6 4

Output:

7 0 8

**Explanation:**

The numbers represented are 342 and 465 (reverse order). Their sum is 807, which in reverse order is 7 0 8.

**Sample Test Case 2:**

Input:

3

2 14 3

3

5 16 4

Output:

7 2 0 8

**Explanation:**

The numbers represented are 3412 and 4615 (reverse order). Their sum is 8027, which in reverse order is 7 2 0 8.

**Constraints:**

The number of nodes in each linked list is in the range [1, 100].

0 <= Node.val <= 9

**Extra Test Cases:**

**Test case 1:**

Input:

1

0

1

0

Output:

0

**Test case 2:**

Input:

2

1 8

1

0

Output:

1 8

**Test case 3:**

Input:

7

9 9 9 9 9 9 9

4

9 9 9 9

Output:

8 9 9 9 0 0 0 1

**Test case 4:**

Input:

2

2 4

3

5 6 7

Output:

7 0 8

**Test case 5:**

Input:

1

0

3

1 2 3

Output:

1 2 3

**2 Lucas and Sorted List**

Lucas has a sorted list of his favorite songs but noticed some duplicates. He wants to remove them to have unique songs in his playlist.

Given a sorted linked list, delete all duplicates such that each element appears only once.

**Input Format**

First line contains input n

Next line contains n elements of LinkedList.

**Output Format**

Print the LL after deleting duplicate elements.

**Sample Test Case 1:**

Input:

3

1 1 2

Output: 1 2

**Explanation:**

The list 1 1 2 has a duplicate of 1. Removing duplicates results in 1 2.

**Sample Test Case 2:**

Input:

4

1 1 2 2

Output:

1 2

**Explanation:**

The list 1 1 2 2 has a duplicate of 1 and 2. Removing duplicates results in 1 2.

**Constraints:**

The number of nodes in the list is in the range [0, 300].

100 <= Node.val <= 100

**Extra Test Cases:**

Input:

5

1 1 2 3 3

Output:

1 2 3

Input:

0

NULL

Output: NULL

Input:

4

1 1 1 1

Output:

1

Input:

5

3 3 2 1 1

Output:

3 2 1

Input:

5

5 5 5 5 6

Output: 5 6

**3 Ella and Songs**

Ella wants to find the common songs between two different playlists represented as linked lists. She needs to find the intersection of these lists.

Given two singly linked lists, find if they intersect and return the intersecting node. If no intersection, return null.

**Input Format**

First line contains input n

second line contains n elements of LinkedListA.

Third line contains input k

next line contains k elements of LinkedListB

**Output Format**

Print the Intersection point of both the lists.

**Sample Test Case:**

**Input**:

5

4 1 8 4 5

6

5 6 1 8 4 5

**Output**:

8

**Explanation**: Both lists intersect at the node with value 8. After this node, both lists share the same nodes.

**Sample Test Case:**

**Input**:

5

4 1 9 4 5

6

5 6 1 8 4 5

**Output**:

4

**Explanation**: Both lists intersect at the node with value 4. After this node, both lists share the same nodes.

**Constraints:**

The number of nodes in both lists is in the range [0, 30000].

100000 <= Node.val <= 100000

**Extra Test Cases:**

**Test Cases 1:**

Input:

5

1 9 1 2 4

3

3 2 4

Output:

2

**Test Cases 2:**

Input:

4

2 6 4 5

2

4 5

Output:

4

**Test Cases 3:**

Input:

3

1 2 3

2

2 3

Output:

2

**Test Cases 4:**

Input:

5

1 2 3 4 5 6

3

4 5 6

Output:

4

**4 Swap Nodes in Pairs**

Mia is organizing pairs of photos for a slideshow. She wants to swap every two adjacent nodes in her linked list.

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed).

**Input Format**

First line contains input n

second line contains n elements of LinkedListA.

**Output Format**

Print the LL after swapping nodes in pairs.

**Sample Test Case 1:**

Input:

4

1 2 3 4

Output:

2 1 4 3

**Explanation**: The pairs (1, 2) and (3, 4) are swapped, resulting in 2 1 4 3.

**Sample Test Case 2:**

Input:

4

11 12 13 14

Output:

12 11 14 13

**Explanation**: The pairs (11, 12) and (13, 14) are swapped, resulting in 12 11 14 13.

**Constraints:**

The number of nodes in the list is in the range [0, 100].

0 <= Node.val <= 100

**Extra Test Cases:**

**Test Cases 1:**

Input:

1

1

Output:

1

**Test Cases 2:**

Input:

1

2

Output:

2

**Test Cases 3:**

Input:

3

1 2 3

Output:

2 1 3

**Test Cases 4:**

Input:

5

4 5 6 7 8

Output:

5 4 7 6 8

**Test Cases 5:**

Input:

5

1 2 3 4 5

Output:

2 1 4 3 5

**5 Delete Node in a Linked List**

Mark is updating his playlist and wants to remove a specific song (node) from the list without having to traverse the entire list.

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

**Input Format**

First line contains n (Number elements)

Second line contains n elements of LinkedListA.

Third line contains the node that you needs to delete.

**Output Format**

Print the new LL after Deleting the elements.

**Sample Test Case 1:**

**Input**:

4

4 5 1 9

5

**Output**:

4 1 9

**Explanation**: Given the node with value 5, it's removed by copying the value of the next node (1) into it and deleting the next node.

**Sample Test Case 2:**

Input:

5

64 36 17 39

36

Output:

64 17 39

**Explanation**: Given the node with value 36, it's removed by copying the value of the next node (17) into it and deleting the next node.

**Constraints:**

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

1000 <= Node.val <= 1000

The node to be deleted is not a tail node in the list.

**Extra Test Cases:**

Input:

4

1 2 3 4

3

Output:

1 2 4

Input:

2

0 1

0

Output: 1

Input:

5

1 2 3 4 5

2

Output:

1 3 4 5

Input:

4

4 5 6 7

5

Output:

4 6 7

Input:

4

7 8 9 10

8

Output:

7 9 10

**6 Oliver Family**

Oliver needs to find the lowest common ancestor (LCA) of two nodes in a family tree represented by a binary tree. The LCA is the deepest node that is an ancestor of both nodes.

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

**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

Next line contains the value of P & Q

**Output Format:**

Print the integer value.

**Sample Test Case 1:**

Input: 10 true 20 true 30 false false false true 40 true 50 false false true 60 true 70 false false false

30 60

Output: 10

**Explanation**: The binary tree is:

10

/ \

20 40

/ / \

30 50 60

/

70

The LCA of nodes 60 and 30 is 10, which is their deepest common ancestor.

**Sample Test Case 2:**

Input: 10 true 20 true 30 false false false true 40 true 50 false false true 60 true 70 false false false

50 60

Output: 40

**Explanation**: The binary tree is:

10

/ \

20 40

/ / \

30 50 60

/

70

The LCA of nodes 60 and 50 is 40, which is their deepest common ancestor.

**Constraints**:

The number of nodes in the tree is in the range [2, 10^5].

10^9 <= Node.val<= 10^9

All Node.val are unique.

p and q are different and both exist in the binary tree.

**Extra Test Cases:**

Input:

15 true 25 true 35 false false false true 45 true 55 false false true 65 true 75 false false false

55 75

Output:

45

Input:

5 true 10 false false true 20 true 30 true 40 false false false false

3010

Output:

20

Input:

1 true 2 true 3 false false true 4 true 5 false false true 6 true 7 false falsefalse

57

Output:

4

Input:

8 true 12 false false true 14 true 16 false false true 18 true 20 false false false

1618

Output:

14

Input:

50 true 30 true 20 false false true 40 true 35 false false true 45 false false true 70 true 60 false falsefalse

35 60

Output: 50

**7 Emma project management**

Emma is reconstructing a project management tree from two lists: one representing the tasks in preorder and the other representing the tasks in inorder. She needs to build the binary tree from these lists.

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and then print Level order Traversal of new Tree

**Input Format:**

First line takes input N

Second Line contains N element of preorder traversal

Third Line contains N element of inorder traversal

**Output Format:**

Print the Level OrderTraversal.

**Sample Test Case 1:**

Input:

4

3 9 20 15 7  
9 3 15 207

Output:

3 9 20 15 7

**Sample Test Case 2:**

Input:

6

0 1 3 4 2 5

3 1 4 0 5 2

Output: 3 4 1 2 0 5

**Constraints:**

The number of nodes in the tree is in the range `[1, 3000]`.

3000 <= Node.val<= 3000

preorder` and `inorder` consist of unique values.

Each value in `preorder` also appears in `inorder`.

preorder` and `inorder` are valid traversals of a binary tree.

**Extra Test Cases:**

Input:

6

0 1 3 4 2 9

3 1 4 0 9 2

Output:

3 4 1 2 0 9

Input:

4

3 9 20 15 7  
9 3 15 207

Output:

3 9 20 15 7

Input:

4

3 9 20 15 17  
9 3 15 2017

Output:

3 9 20 15 17

Input:

4

3 91 20 15 17  
91 3 15 2017

Output:

3 91 20 15 17

**8 Lucas Search Engine**

Lucas is managing a search engine and wants to find the k-th smallest keyword in the binary search tree of keywords. He needs to efficiently retrieve the k-th smallest element.

Given the root of a binary search tree, and an integer `k`, return the `k`-th smallest value (1-indexed) of all the values of the nodes in the tree.

**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

Next line contains value of k.

**Output Format:**

Print Integer value.

**Sample Test Case 1:**

Input:

8 true 3 true 1 false false true 6 false false true 10 true 9 false false true 14 false false

1

Output: 1

Explanation: The binary search tree is:

8

/ \

3 10

/ \ \

1 6 14

\

9

The 1st smallest element is 1.

**Sample Test Case 2:**

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

2

Output:

7

Explanation: The binary search tree is:

15

/ \

10 20

/ \ / \

7 12 17 25

/ \ \

5 9 13

/

11

The 2nd smallest element is 7.

**Constraints:**

The number of nodes in the tree is in the range `[1, 10000]`.

0 <= Node.val<= 10^4

All the values of the nodes are unique.

1 <= k <= number of nodes in the BST.

**Extra Test Cases:**

Input:

8 true 3 true 1 false false true 6 false false true 10 true 9 false false true 14 false false

1

Output: 1

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

2

Output: 7

Input:

5 true 3 true 1 false false true 4 false false true 8 true 7 true 6 false false true 9 false false

4

Output:

5

Input:

10 true 5 true 3 true 1 false false true 4 false false true 7 true 6 false false true 8 false false

3

Output:

4

**9 Alex and his company**

Alex is creating a visual representation of the company's hierarchy. He needs to determine the depth of the tree structure representing the organization to properly allocate space for each level.

Given the root of a binary tree, find its maximum depth. The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

**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 Integer value.

**Sample Test Case 1:**

Input:

8 true 3 true 1 false false true 6 false false true 10 true 9 false false true 14 false false

Output:

4

Explanation: The binary search tree is:

8

/ \

3 10

/ \ \

1 6 14

\

9

**Sample Test Case 2:**

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output:

6

Explanation: The binary search tree is:

15

/ \

10 20

/ \ / \

7 12 17 25

/ \ \

5 9 13

/

11

**Constraints:**

The number of nodes in the tree is in the range `[0, 10000]`.

100 <= Node.val<= 100

**Extra Test Cases:**

Input:

8 true 3 true 1 false false true 6 false false true 10 true 9 false false true 14 false false

Output:

4

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output:

5

Input:

0 false false

Output:

1

Input:

15 true 10 true 7 true 5 false false true 9 false false true 12 true 11 false false true 13 false false true 20 true 17 false false true 25 false false

Output:

6

Input:

10 true 5 true 3 true 1 false false true 4 false false true 7 true 6 false false true 8 false false

Output:

5

**10. Implement Queue using Stacks**

Emily wants to build a queue system for handling customer service requests. However, she only has stacks available. She needs to implement a queue using two stacks.

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support `push`, `pop`, `peek`, and `empty` operations.

**Sample Test Case 1:**

**Input:**

Queue queue = new Queue();

queue.push(1);

queue.push(2);

queue.peek(); // returns 1

queue.pop(); // returns 1

queue.empty(); // returns false

**Explanation**: After pushing 1 and 2 to the queue, peeking returns 1 (the front element), popping returns 1 (removes the front element), and checking if the queue is empty returns false.

**Sample Test Case 2:**

**Input:**

Queue queue = new Queue();

queue.push(1);

queue.push(2);

queue.push(3);

queue.pop(); // returns 1

queue.peek(); // returns 2

queue.pop(); // returns 2

queue.empty(); // returns false

**Explanation**: After pushing 1 and 2 and 3 to the queue, pop() return 1 (removes the front element), peeking returns 2 (the front element), popping returns 2 (removes the front element), and checking if the queue is empty returns false because 3 is present there.

**Constraints:**

You must use only standard stack operations: `push`, `pop`, `top`, and `empty`.

Assume all operations are valid (e.g., no pop or peek operations will be called on an empty queue).

**Extra Test Cases:**

Input:

Queue queue = new Queue();

queue.push(1);

queue.push(2);

queue.push(3);

queue.pop(); // returns 1

queue.peek(); // returns 2

queue.pop(); // returns 2

queue.empty(); // returns false

Input:

Queue queue = new Queue();

queue.push(1);

queue.push(2);

queue.pop(); // returns 1

queue.pop(); // returns 2

queue.empty(); // returns true

Input:

Queue queue = new Queue();

queue.push(1);

queue.peek(); // returns 1

queue.pop(); // returns 1

queue.empty(); // returns true

Input:

Queue queue = new Queue();

queue.empty(); // returns true

Input:

Queue queue = new Queue();

queue.push(1);

queue.push(2);

queue.push(3);

queue.push(4);

queue.pop(); // returns 1

queue.pop(); // returns 2

queue.peek(); // returns 3

**Hard**

**1 Tom and his LL**

Given a singly linked list, reverse every alternate k nodes in the list starting from the beginning. For example, if k = 3, reverse the nodes at positions 1, 4, 7, etc.

**Explanation:**

In each test case, the linked list is modified by reversing every alternate group of k nodes starting from the beginning.

The solution should efficiently handle various edge cases, including lists of different lengths and values of k.

Ensure that the implementation meets the constraints of reversing nodes in-place and not modifying node values.

**Constraints:**

You must reverse the nodes in-place without using extra memory.

You cannot modify the values in the nodes, only the nodes themselves may be changed.

Assume k is a positive integer and the list always has at least one node.

0 <= Node.val <= 1000

1 <= K <= N <=2000

**Input Format:**

The first line of input contains an integer N,the Number of Elements in the LinkedList.

The next lines contains the space separated n node data values in order.

Last Line Contains an integer K,the group of the node to reverse.

**Output Format:**

Print the modified linked list after reversing every alternate k nodes.

**Sample Test Cases:**

Test case 1:

Linked List:

8

1 2 3 4 5 6 7 8

3

Output:3 2 1 4 5 6 8 7

Test case 2:

9

1 2 3 4 5 6 7 8 9

2

Output : 2 1 3 4 6 5 7 8 9

Input:

9

10 20 30 40 50 60 70 80 90

4

Output:

40 30 20 10 50 60 70 80 90

Input:

5

1 2 3 4 5

5

Output: 5 4 3 2 1

Input:

9

1 2 3 4 5 6 7 8 9

1

Output: 1 2 3 4 5 6 7 8 9

**2 Alice's Playlist**

Alice wants to reorganize her playlist by grouping every k songs and reversing the order within each group.

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list. If the number of nodes is not a multiple of k, leave the remaining nodes as they are.

**Input Format:**

The first line of input contains an integer N,the Number of Elements in the LinkedList.

The next lines contains the space separated n node data values in order.

Last Line Contains an integer K,the group of the node to reverse.

**Output Format:**

Print the space separated n node data values of the LinkedList in order after reversing .

**Sample Test Case 1:**

Input:

5

1 2 3 4 5

3

Output: 3 2 1 4 5

**Explanation**: The first group of 3 nodes (1, 2, 3) is reversed to 3 -> 2 -> 1. The remaining nodes (4, 5) are left as is.

**Sample Test Case 2:**

Input:

5

1 2 3 4 5

2

Output: 2 1 4 3 5

**Explanation**: The first group of 3 nodes (1, 2) is reversed to 2 -> 1. The remaining nodes (3, 4) is reversed to 4 -> 3 and 5 are left as it is.

**Constraints:**

0 <= Node.val <= 1000

1 <= K <= N <=2000

**Extra Test Cases:**

Input:

8

1 2 3 4 5 6 7 8

2

Output: 2 1 4 3 6 5 8 7

Input:

6

1 2 3 4 5 6

3

Output: 3 2 1 6 5 4

Input:

5

1 2 3 4 5

1

Output: 1 2 3 4 5

Input:

3

1 2 3

4

Output: 1 2 3

Input:

4

1 2 3 4

2

Output: 2 1 4 3

**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