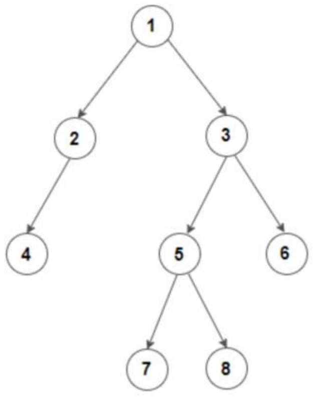
Q1. Write an efficient function to construct a binary tree from the given inorder and preorder sequence. For example,

Input:

Inorder Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }

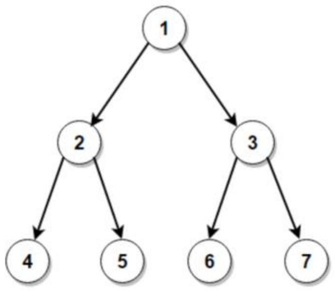
Preorder Traversal: { 1, 2, 4, 3, 5, 7, 8, 6 }

Output: Below binary tree



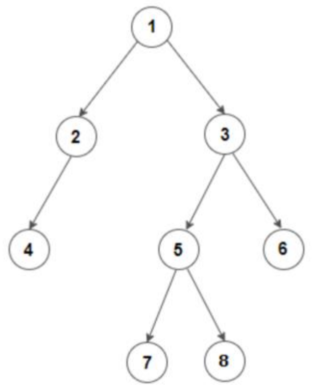
Q2. Write an efficient function to construct a binary tree from the given inorder and level order sequence. For example,

Input:
Inorder Traversal : { 4, 2, 5, 1, 6, 3, 7 }
level order traversal : { 1, 2, 3, 4, 5, 6, 7 }
Output: Below binary tree



Q3. Write an efficient algorithm to find postorder traversal on a given binary tree from its inorder and preorder sequence.

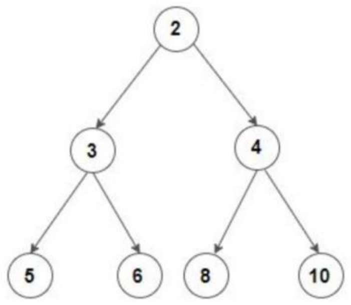
For example, consider the following tree:



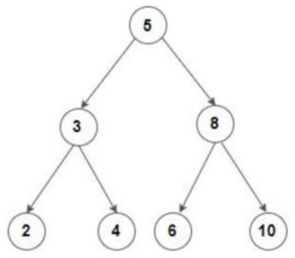
Input:
Inorder traversal is { 4, 2, 1, 7, 5, 8, 3, 6 }
Preorder traversal is { 1, 2, 4, 3, 5, 7, 8, 6 }
Output:
Postorder traversal is 4 2 7 8 5 6 3 1

Q4. Given a binary tree, check if it is a minimum heap or not. In order words, the binary tree must be a complete binary tree where each node has a higher value than its parent's value.

For example, the following binary tree is a minimum heap:

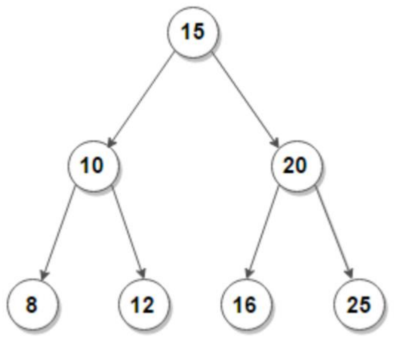


On the other hand, the following binary tree is not a minimum heap:



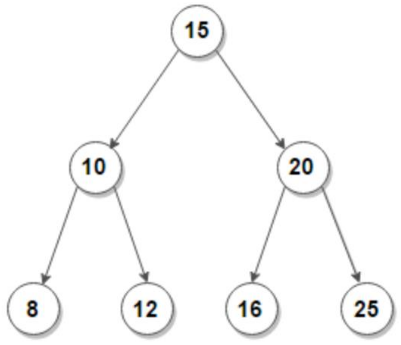
Q5. Given a distinct sequence of keys representing the postorder traversal of a binary search tree, construct a BST from it.

For example, the following BST should be constructed for postorder traversal {8, 12, 10, 16, 25, 20, 15}:



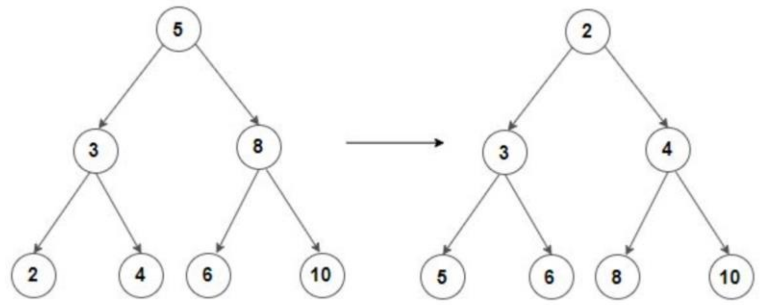
Q6. Given a BST and a positive number k, find the k'th largest node in the BST.

For example, consider the following binary search tree. If k = 2, the k'th largest node is 20.



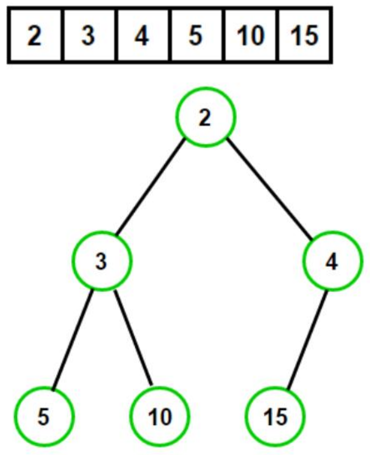
Q7. Given a binary search tree (BST), efficiently convert it into a minimum heap. In order words, convert a binary search tree into a complete binary tree where each node has a higher value than its parent's value.

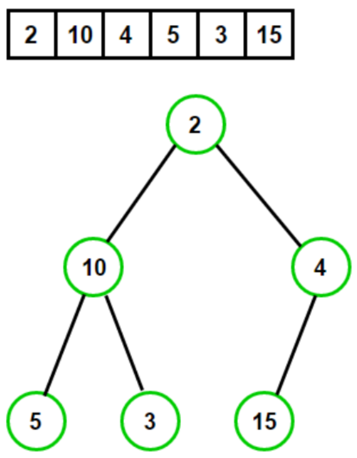
For example, the solution should convert the BST on the left into the binary tree on the right, or any other binary tree with the same set of keys that satisfies the structural and heap-ordering property of minimum heap data structure.



Q8. Given an integer array, check if it represents minimum heap or not.

For example, the first array represents a minimum heap, but the second array isn't as it violate the heap property.





Q9. Given k sorted linked lists, merge them into a single list in increasing order using heap.

For example,

Input: k = 3

List 1: 1 —> 5 —> 7 —> NULL

List 2: 2 —> 3 —> 6 —> 9 —> NULL

List 3: 4 —> 8 —> 10 —> NULL

Output: 1 —> 2 —> 3 —> 4 —> 5 —> 6 —> 7 —> 8 —> 9 —> 10 —> NULL

Idea: The idea is to construct a min-heap of size k and insert each list's first node into it. Then pop the root node (having minimum value) from the heap and insert the next node from the "same" list as the popped node. We repeat this process until the heap is exhausted.

Q10. Given two arrays apples[] and days[] representing the count of apples an apple tree produces and the number of days these apples are edible from the ith day respectively, the task is to find the maximum number of apples a person can eat if the person can eat at most one apple in a day using priority queues.

Examples

Input: apples[] = { 1, 2, 3, 5, 2 }, days[] = { 3, 2, 1, 4, 2 }

Output: 7

Explanation:

On 1st day, person eats the apple produced by apple tree on the 1st day.

On 2nd day, person eats the apple produced by apple tree on the 2nd day.

On 3rd day, person eats the apple produced by apple tree on the 2nd day.

On 4th to 7th day, person eats the apple produced by apple tree on the 4th day.

Input: apples[] = { 3, 0, 0, 0, 0, 2 }, days[] = { 3, 0, 0, 0, 0, 2 }

Output: 5