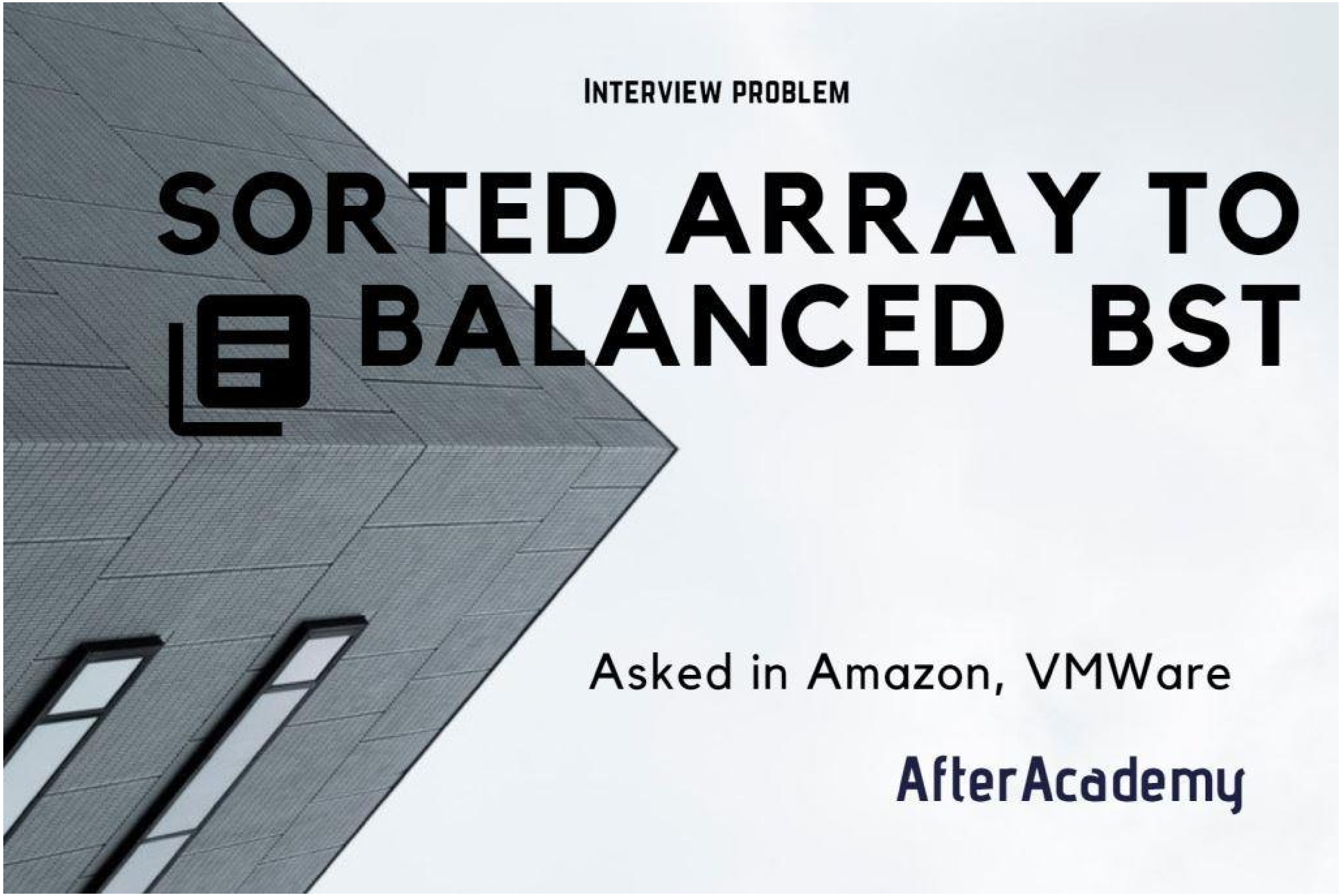
AfterAcademy

Admin AfterAcademy
24 Dec 2019

Sorted Array to balanced BST


INTERVIEW PROBLEM
SORTED ARRAY TO BALANCED BST
Asked in Amazon, VMWare
AfterAcademy

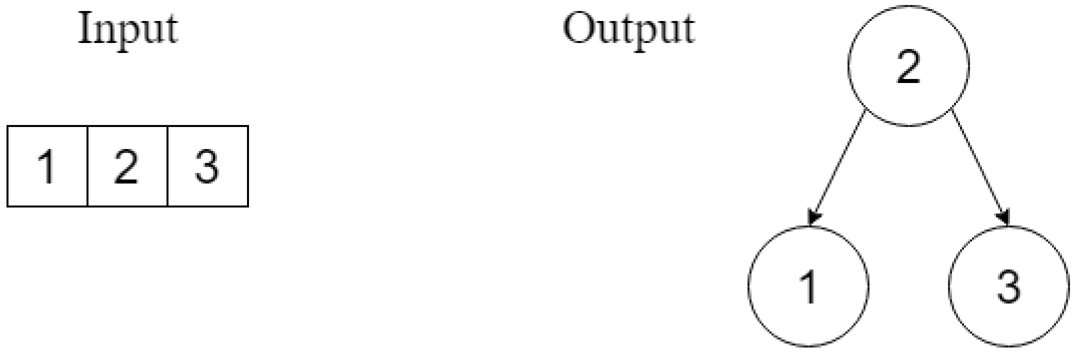
Asked in: VMWare, Amazon

Difficulty: Easy

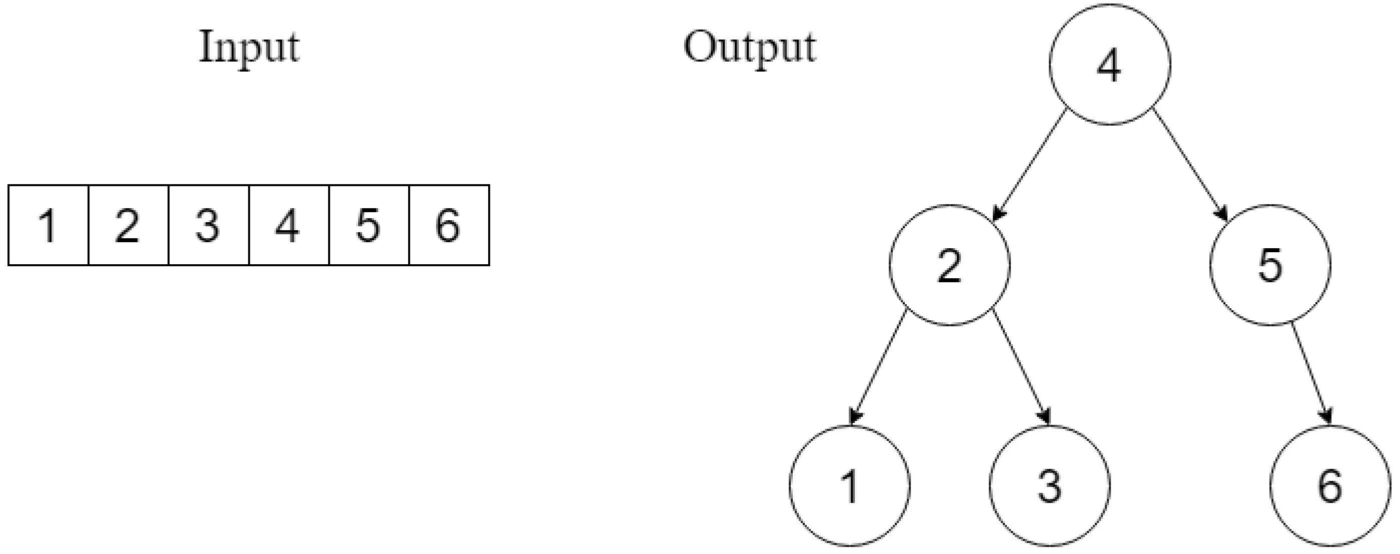
Understanding the problem

Given an array where elements are sorted in ascending order, convert it to a height balanced BST. For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Example 1



Example 2



The node structure for the BST returned from your function will be—

```
class TreeNode
{
    int val
    TreeNode left
    TreeNode right
}
```

Recursive and iterative solutions

We will be discussing two approaches to solve this problem:

1. **Using divide & conquer approach:** Choose the middle element as the root and recursively build balanced BST for the left and right part of the array.
2. **Iterative solution:** Similar to level order traversal, keep track of middle elements of the array for each level using the stack.

1. Recursive Solution

Here we use the sorted property of the array to ensure the construction of balanced BST where we divide the array into two equal parts and assign the mid value as a root node. To be in alignment with the definition of a Binary Search Tree, the elements in the array to the left of the mid value, would contribute to the left subtree while the elements in the array to the right of the mid value, would contribute to the right subtree.

The steps to be followed are :

1. Get the Middle of the array and make it root.
2. Recursively do the steps for the left half and right half.

- Get the middle of the left half and make it the left child of the root.
- Get the middle of the right half and make it the right child of the.

```
TreeNode sortedArrayToBST_helper(int A[], int start, int end)
{
    // continue while this branch has values to process
    if(start > end)
       return NULL
    // Get the middle element and make it root
    int mid = start + (end - start)/2
    TreeNode root = new TreeNode(A[mid])
    // Recursively construct the left subtree
    // and make it left child of root
    root.left = sortedArrayToBST_helper(A, start, mid - 1)
    // Recursively construct the right subtree
    // and make it right child of root
    root.right = sortedArrayToBST_helper(A, mid + 1, end)
    return root
}

TreeNode sortedArrayToBST(int A[], int n)
{
    TreeNode root = sortedArrayToBST_helper(A, 0, n)
    return root
}
```

Complexity Analysis

In the above approach, we are traversing each index of the array only once. **Time complexity**: **O(n)**, where n is the length of the array.

Space Complexity of this algorithm is proportional to the maximum depth of recursion tree generated which is equal to the height of the tree (h). Here the tree will be balanced, So the maximum height will be log(n) where n is the length of the array.

Space complexity: **O(h)** for recursion call stack, where h is the height of the tree.

Critical ideas to think

- How the height of the tree is balanced using the above approach?
- Can you write the recurrence relation for this approach?
- Explore different types of balanced BST structure like AVL tree, red black tree, 2-3 tree etc.

2. Iterative Solution

- The recursive solution is very intuitive. To reformat it as iterative, the overall idea is to create a stack that keeps track of tuples of information of child nodes we need to create.
- Each tuple keeps track of the child's parent and the side of the parent that the child will become.
- We only push child tuples to the stack after their parents are created, the process will create the children until we reach the base case, whereby that branch has exhausted its corresponding chunk of the original nums.

Pseudo Code

```
struct T {
   int low_idx
   int high_idx
   TreeNode node
   Tree(int low, int high, TreeNode _node) {
       low_idx = low
       high_idx = high
       node = _node
   }
}

TreeNode sortedArrayToBST(int A[], int n) {
    // return if the length of array is 0
    if (n is 0)
        return null
    // create stack and push the node with middle element of array
    stack S
    TreeNode root = new TreeNode(A[(n - 1)/2])
    T tree = new T(0, n - 1, root)
    S.push(tree)
    while (S is not empty) {
        // pop the top node and assign the left and right child to it
        T tmp = S.top()
        S.pop()
        int mid = tmp.low_idx + (tmp.high_idx - tmp.low_idx) / 2
        if (tmp.low_idx < mid) {
            TreeNode node = new TreeNode(A[tmp.low_idx + (mid - 1 - tmp.low_idx) / 2])
            tmp.node.left_idx = node
            tree = new T(tmp.low_idx, mid - 1, node)
            S.push(tree)
        }
        if (mid < tmp.high_idx) {
            TreeNode node = new TreeNode(A[mid + 1 + (tmp.high_idx - mid - 1)/2])
            tmp.node.right_idx = node
            tree = new T(mid + 1, tmp.high_idx, node)
            S.push(tree)
        }
    }
    return root
}
```

Complexity Analysis

For each node in the tree, we are performing push() and pop() operation only once.

Total no of stack operations = 2n **(Think!)**

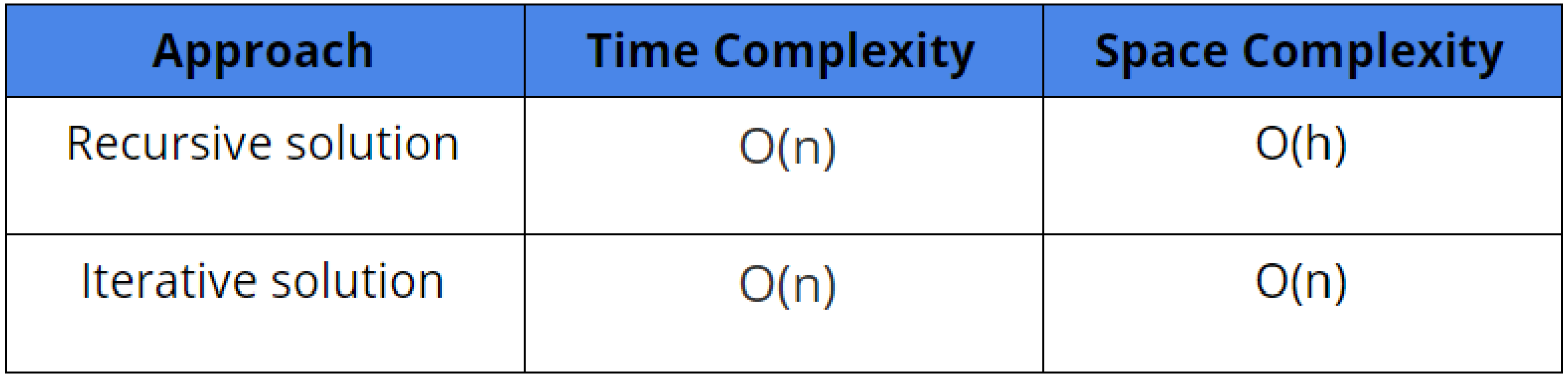
Time complexity: O(n), where n is the total number of nodes.

Space Complexity: O(n)**(Think!)**

Critical Ideas to think

- Visualize the stack operations via simple examples.
- Why the stack is augmented with low and high index values?
- How we are deciding a node to be the left or right child of the root?

Comparison of different approaches



Approach	Time Complexity	Space Complexity
Recursive solution	O(n)	O(h)
Iterative solution	O(n)	O(n)

Suggested Problems to Solve

- Sorted Linked List to Balanced BST
- Sorted order printing of a given array that represents a BST
- Find kth smallest element in BST
- Determine if a tree is a height-balanced tree or not
- Construct a complete binary tree from the given array

Happy Coding! Enjoy Algorithms!

AfterAcademy Data Structure And Algorithms Online Course—Admissions Open

NEW

Google Android Developer Interview

Google Android Engineer Interview Process and Preparation.

CHECK NOW

Share this blog and spread the knowledge

SHARE ON FACEBOOK SHARE ON TWITTER

SHARE ON LINKEDIN SHARE ON TELEGRAM

SHARE ON REDDIT SHARE ON WHATSAPP

1 Comment Sort by Oldest

Add a comment...

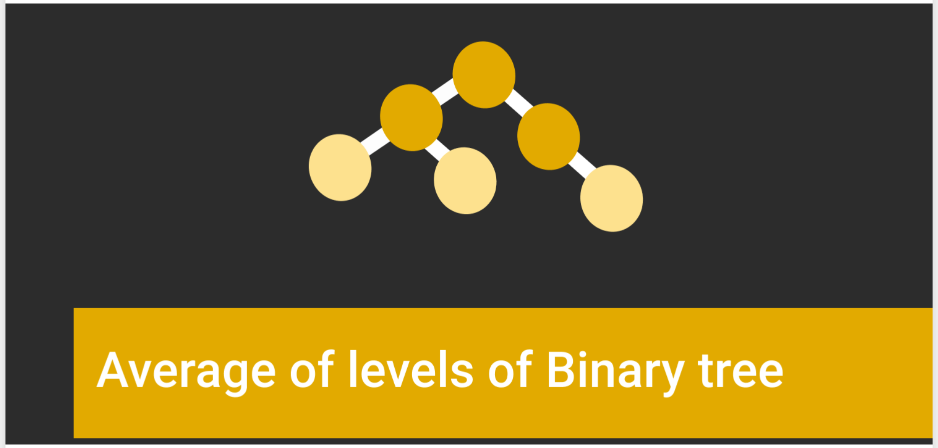
Bogdan Nadgob

If you apply the recursive algorithm for [1, 2, 3, 4, 5, 6] you cannot get the BST from above.
Also for the iterative approach:
1. what are left_idx and right_idx? They are not defined in the TreeNode class.
2. The constructor name for the T structure should not be Tree.

Like · Reply · 1y

Facebook Comments Plugin

Recommended for You



Average of Levels in Binary Tree

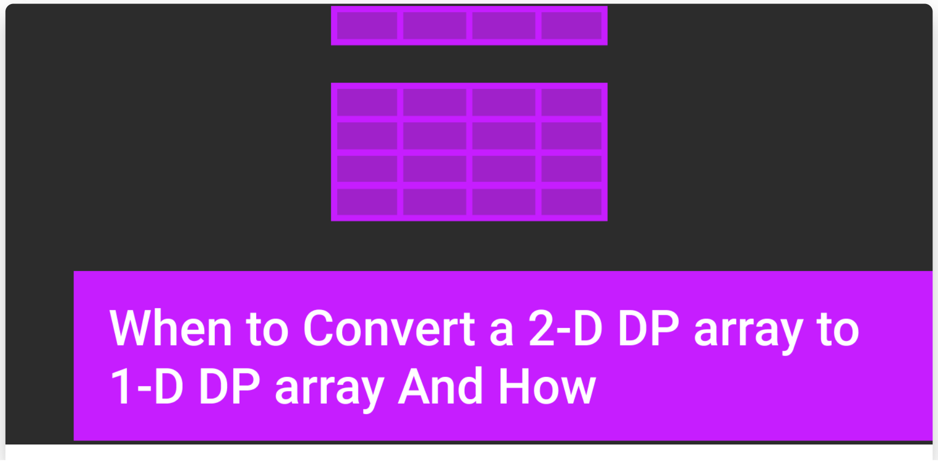
Given a binary tree, write a program to return the average value of the nodes on each level in the form of an array. The range of the node's value is in the range of 32-bit signed integer.

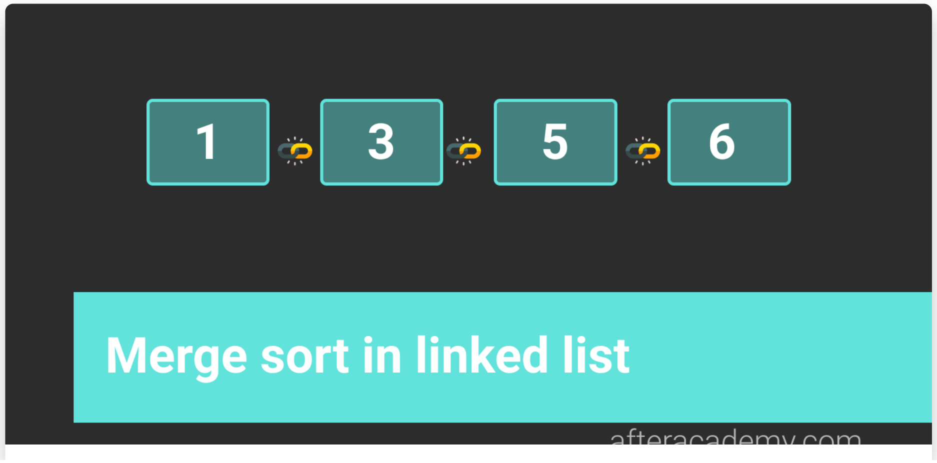
Admin AfterAcademy
5 Nov 2020

Recursive Insertion Sort

Write a program for the recursive implementation of Insertion Sort. Insertion Sort is used to sort a given array. This problem will sharpen your recursion skills.

Admin AfterAcademy
23 Sep 2020





When to Convert a 2-D DP array to 1-D DP array And How?

In which situation 2 dimensional DP can be dropped to 1 dimension? Is there any principle or regular pattern? This is a very important question when it comes to optimization of DP arrays. Let's find out.

Admin AfterAcademy
16 Sep 2020

Sort List - Merge Sort

Sort a linked list using Merge Sort. This is a very famous interview problem that demonstrates the concept of recursion. This problem is quite similar to Merge Sort in Arrays.

Admin AfterAcademy
12 Sep 2020

Largest Element In An Array

Given an array arr[] of size n , write a program to find the largest element in it. To find the largest element we can just traverse the array in one pass and find the largest element by maintaining a max variable.

Admin AfterAcademy
4 Sep 2020

Merge Two BST

Given two binary search trees with root nodes as tree1 and tree2 of size n and m, write a program to return an array of integers that contains all the elements of tree1 and tree2 in non-decreasing order. The expected time complexity is O(m+n).

Admin AfterAcademy
1 Sep 2020

Our Learners Work At

makemytrip zomato Paytm GOJEK Directi Intelligent People. Uncommon Ideas.

hike Microsoft Rakuten zalando SWIGGY

AfterAcademy

Stay up to date. Follow us on