Binary Tree - Complete Notes

Table of Contents

Introduction

A **Binary Tree** is a hierarchical data structure where each node has at most two children, referred to as the left child and right child.

Basic Structure

```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;

    TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

Types of Binary Trees

1. Perfect Binary Tree

A binary tree is **perfect** if:

- All internal nodes have exactly two children
- All leaf nodes are at the same level
- The tree is completely filled up to the nth level

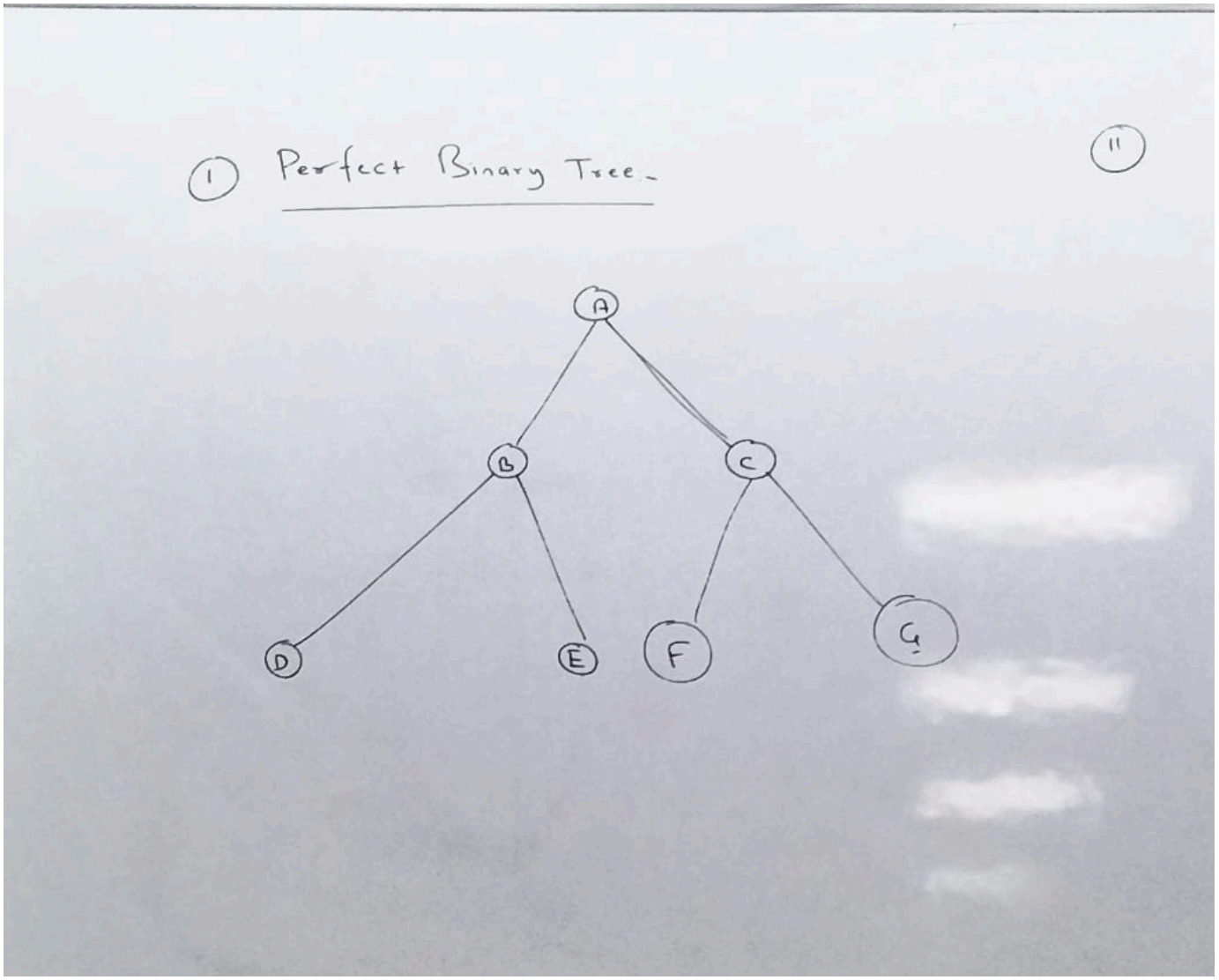
Properties:

- Total nodes = 2^(h+1) - 1, where h is height
- Leaf nodes = 2^h

- Height = $\log_2(n+1) - 1$

Example:

```
      1
    /   \
   2     3
  / \   / \
 4   5 6   7
```



2. Full Binary Tree

A binary tree is **full** if every node has either:

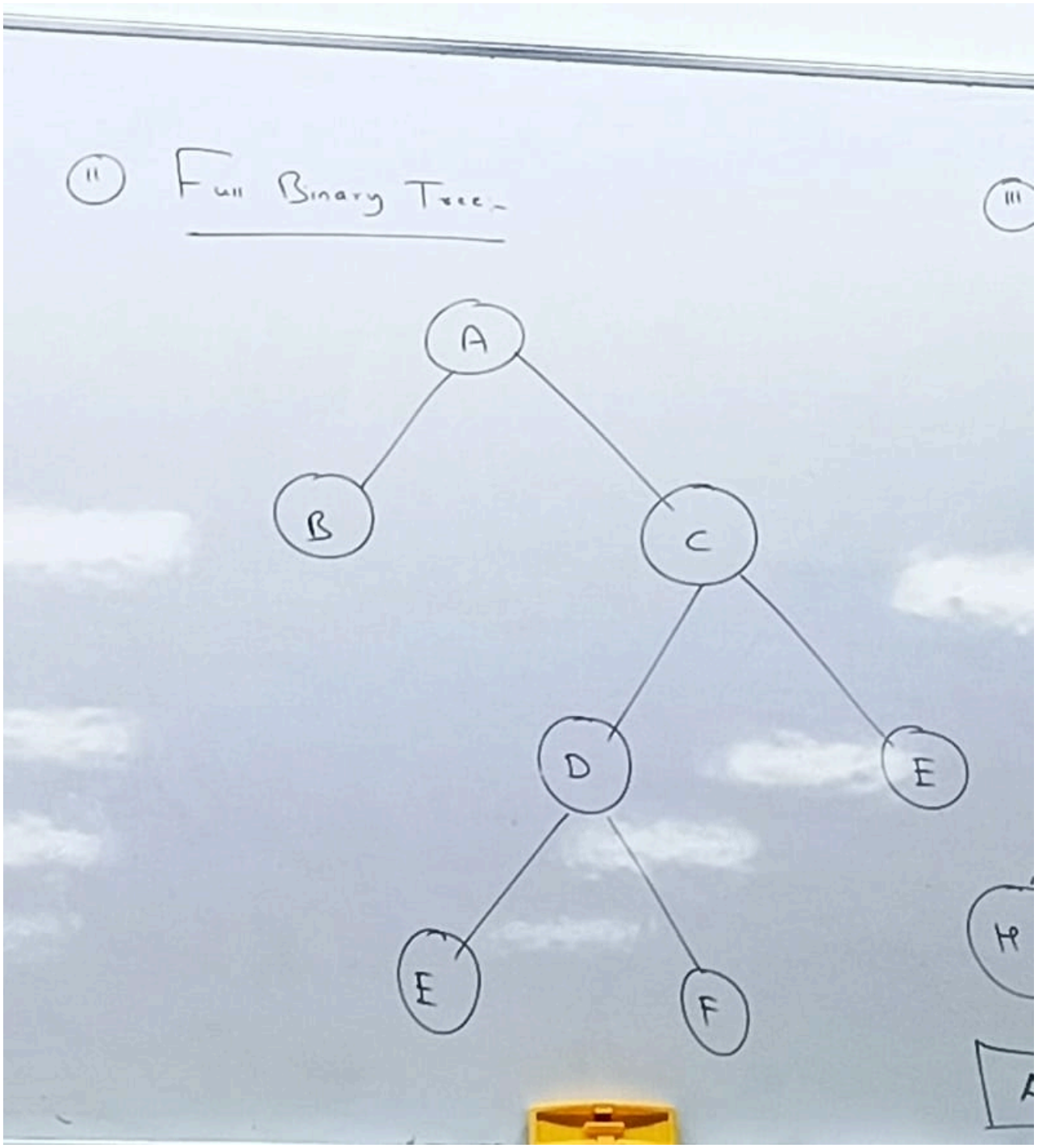
- Zero children (leaf node), OR
- Two children (internal node)

Properties:

- If there are n nodes, then there are (n+1)/2 leaf nodes
- Number of internal nodes = (n-1)/2

Example:

```
      1
    /   \
   2     3
  / \
 4   5
```



3. Complete Binary Tree

A binary tree is **complete** if:

- All levels are completely filled except possibly the last level
- The last level is filled from left to right

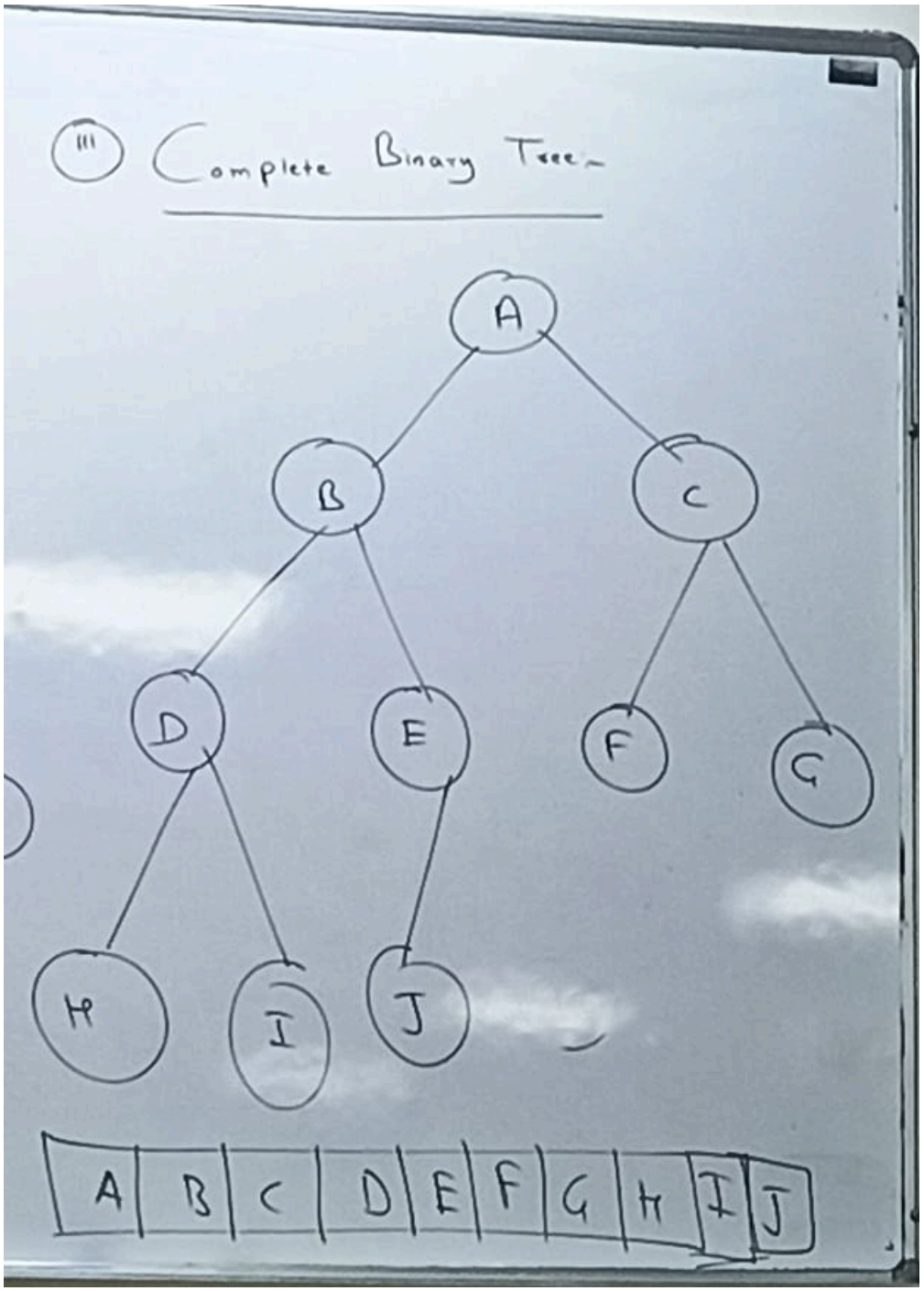
- Used in heap data structures

Properties:

- Height = $\lfloor \log_2(n) \rfloor$
- Efficiently represented using arrays
- Parent of node at index i is at $\lfloor (i-1)/2 \rfloor$
- Left child at 2i + 1, Right child at 2i + 2

Example:

```
      1
    / \
   2   3
  / \  /
 4  5 6
```



4. Degenerate (Skewed) Binary Tree

Every parent node has only one child. Performance degrades to O(n) like a linked list.

Example (Left Skewed):

```
    1
   /
  2
 /
3
```

5. Balanced Binary Tree

A binary tree where the height difference between left and right subtrees is at most 1 for every node. Examples: AVL Tree, Red-Black Tree.

Tree Traversal Methods

1. Inorder Traversal (Left → Root → Right)

Visits nodes in ascending order for BST.

```
public void inorderTraversal(TreeNode root) {
    if (root == null) return;

    inorderTraversal(root.left);
    System.out.print(root.data + " ");
    inorderTraversal(root.right);
}
```

Time Complexity: O(n) **Space Complexity:** O(h) - recursion stack, where h is height

---

2. Preorder Traversal (Root → Left → Right)

Used to create a copy of the tree or get prefix expression.

```
public void preorderTraversal(TreeNode root) {
    if (root == null) return;

    System.out.print(root.data + " ");
    preorderTraversal(root.left);
    preorderTraversal(root.right);
}
```

Time Complexity: O(n) **Space Complexity:** O(h)

---

3. Postorder Traversal (Left → Right → Root)

Used to delete the tree or get postfix expression.

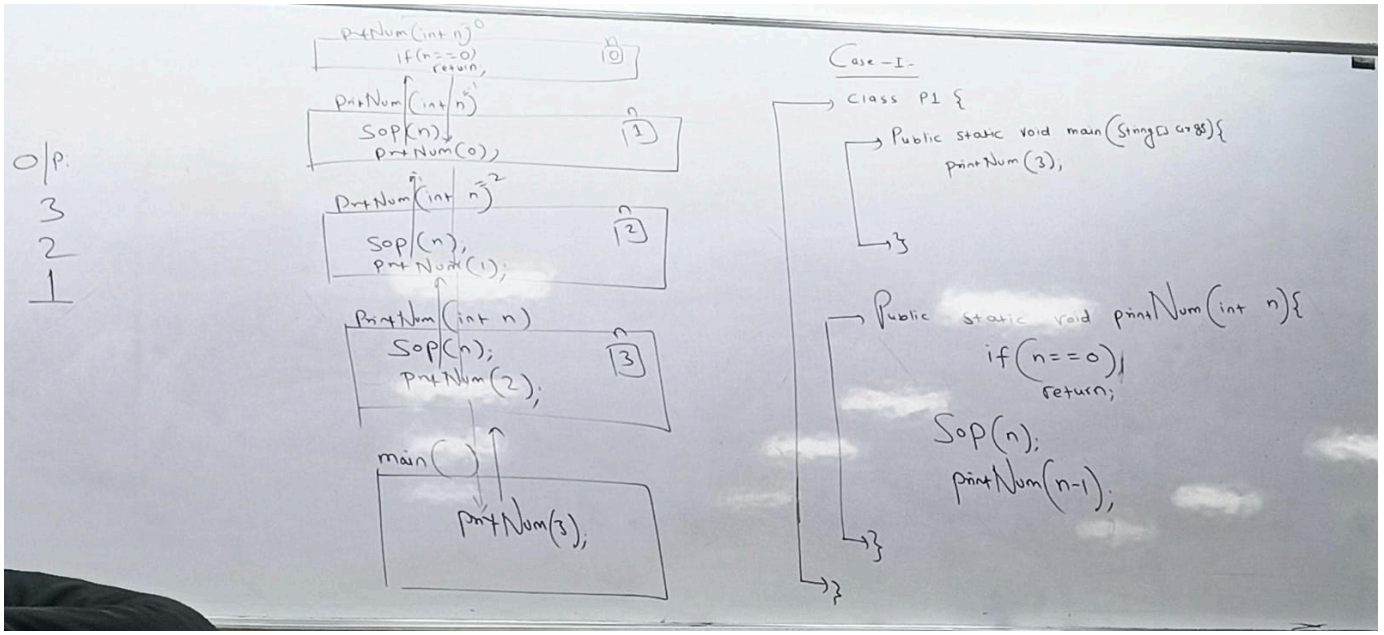
```
public void postorderTraversal(TreeNode root) {
    if (root == null) return;

    postorderTraversal(root.left);
    postorderTraversal(root.right);
    System.out.print(root.data + " ");
}
```

Time Complexity: O(n) **Space Complexity:** O(h)

Traversal Cases Summary

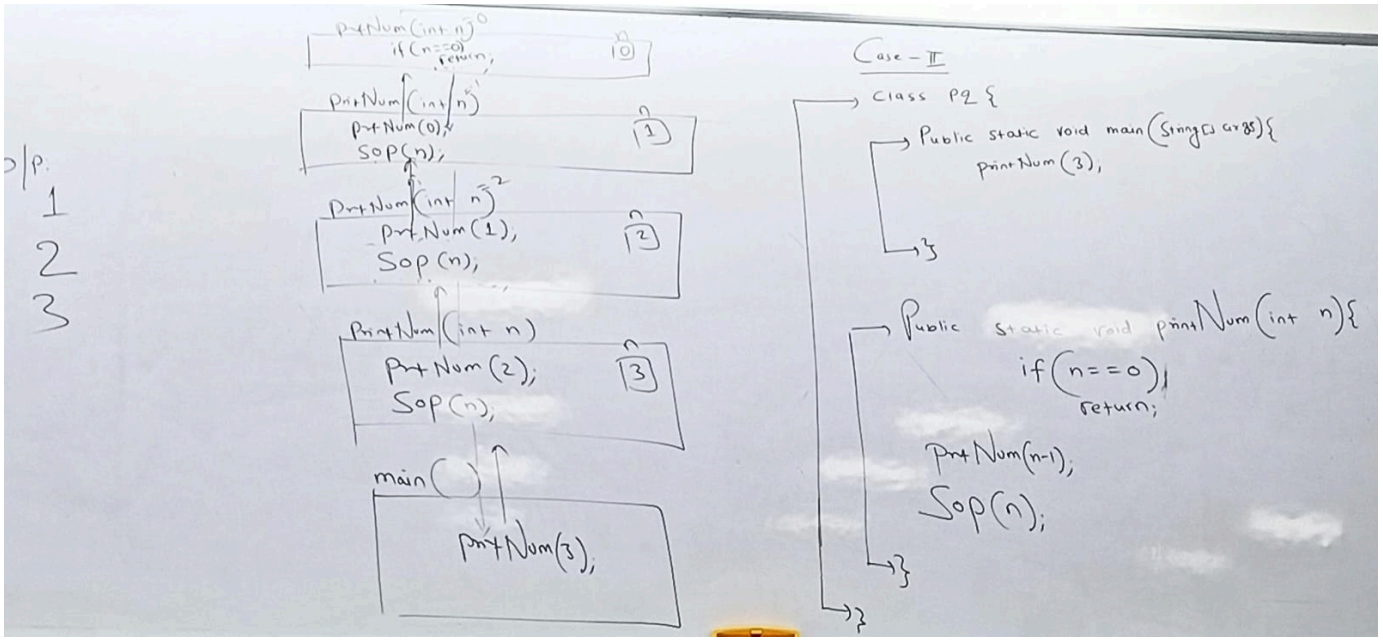
Case 1: Print First, Then Recurse (Preorder)

```
void case1(TreeNode root) {
    if (root == null) return;
    System.out.print(root.data + " ");  // Print first
    case1(root.left);                    // Then recurse left
    case1(root.right);                   // Then recurse right
}
```



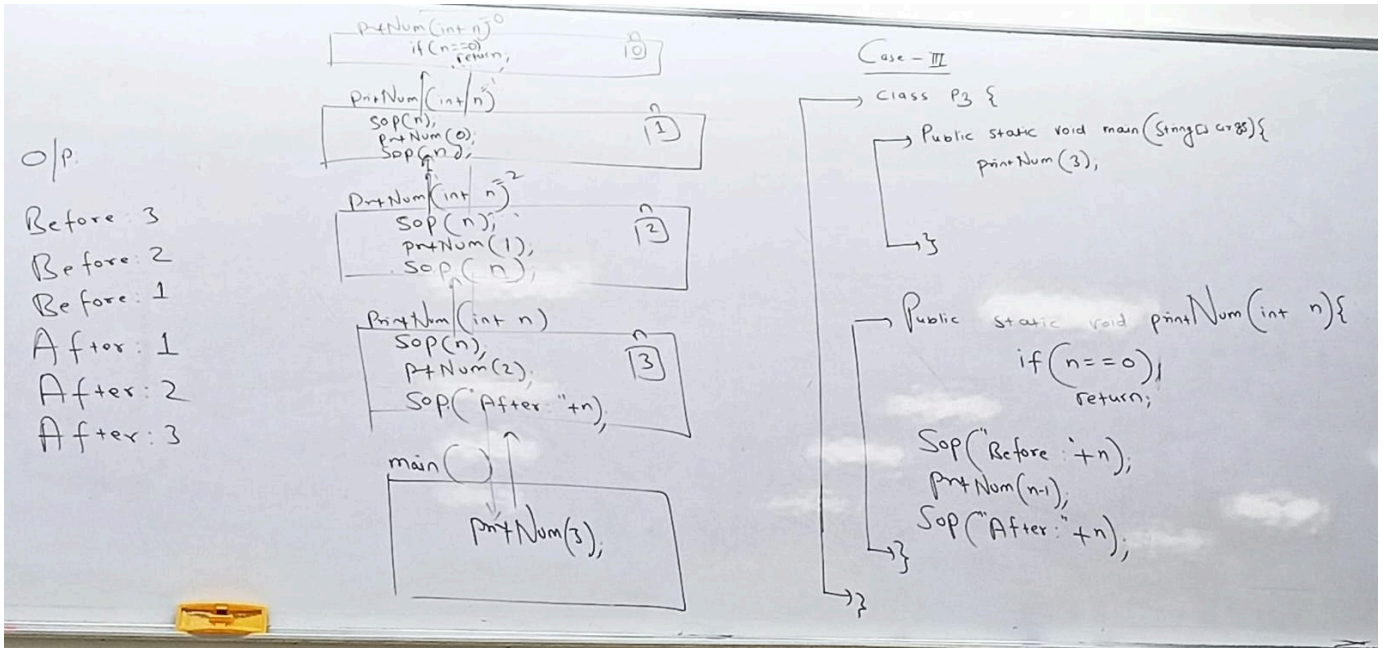
Case 2: Recurse First, Then Print (Postorder)

```
void case2(TreeNode root) {
    if (root == null) return;
    case2(root.left);                    // Recurse left first
    case2(root.right);                   // Then recurse right
    System.out.print(root.data + " ");  // Print last
}
```



Case 3: Print Before and After Recursion (Inorder with extras)

```
void case3(TreeNode root) {
    if (root == null) return;
    System.out.print(root.data + " ");  // Print before
    case3(root.left);                    // Recurse left
    case3(root.right);                   // Recurse right
    System.out.print(root.data + " ");  // Print after
}
```



Binary Search Tree (BST)

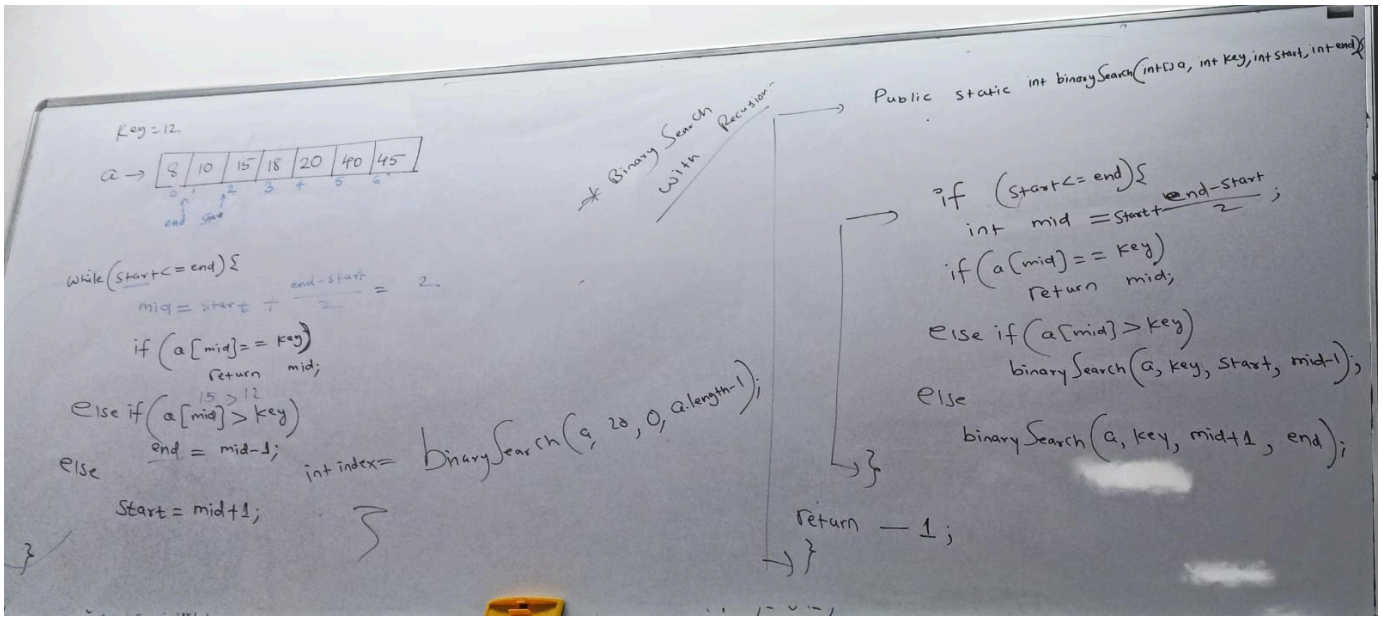
A BST is a binary tree where:

- Left subtree contains only nodes with keys less than the node's key
- Right subtree contains only nodes with keys greater than the node's key

- Both subtrees are also BSTs

Search in BST (Recursive)

```
public TreeNode search(TreeNode root, int key) {
    // Base case: root is null or key is present at root
    if (root == null || root.data == key) {
        return root;
    }

    // Key is greater than root's key
    if (key > root.data) {
        return search(root.right, key);
    }

    // Key is smaller than root's key
    return search(root.left, key);
}
```

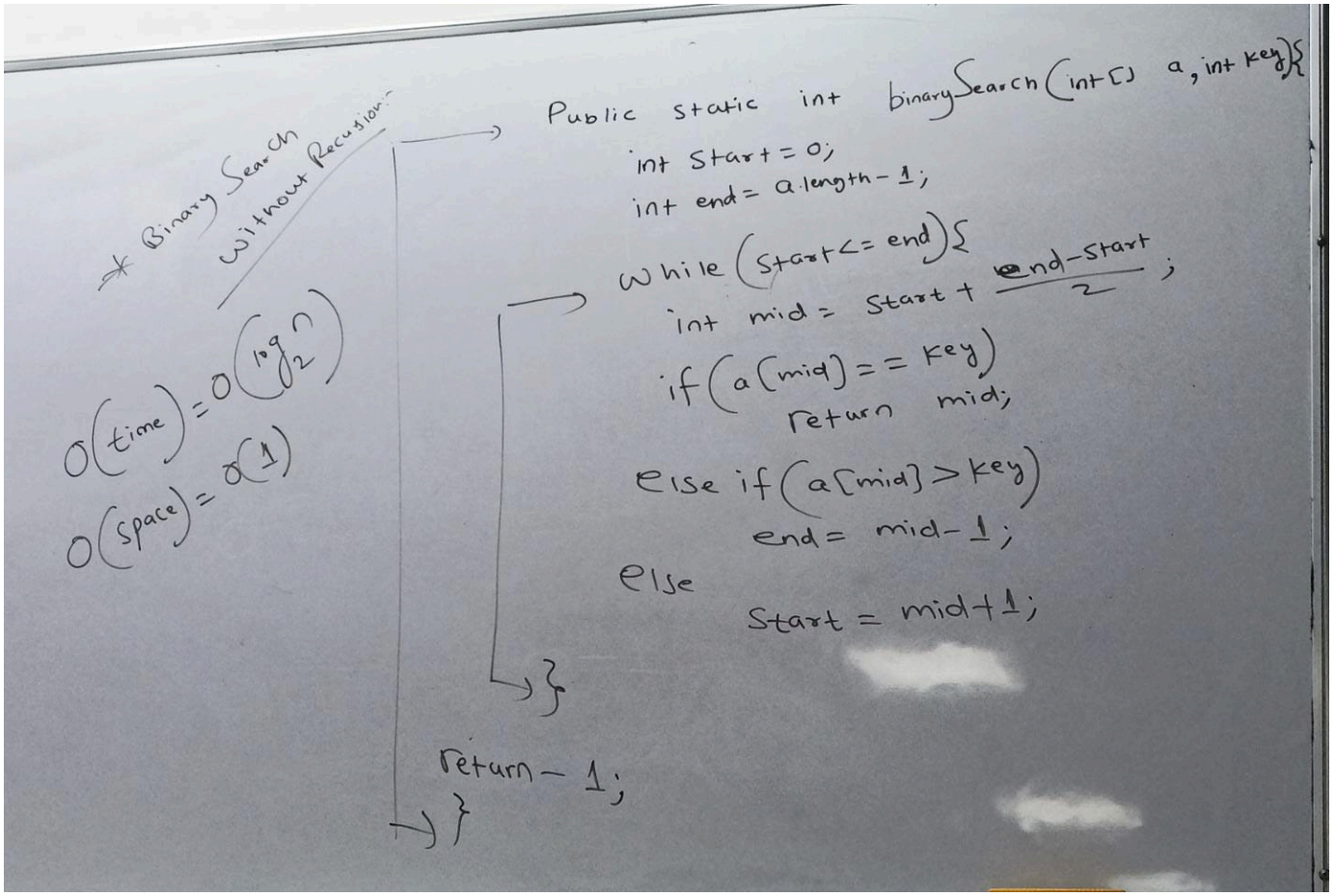


Time Complexity: O(h) - O(log n) for balanced, O(n) for skewed **Space Complexity:** O(h) - recursion stack

---

Search in BST (Iterative)

```java
public TreeNode searchIterative(TreeNode root, int key) {
    TreeNode current = root;

    while (current != null && current.data != key) {
        if (key > current.data) {
            current = current.right;
        } else {
            current = current.left;
        }
    }

    return current;
}
```



Time Complexity: O(h) **Space Complexity:** O(1)

Complexity Analysis

Time Complexities Summary

Operation	BST (Balanced)	BST (Skewed)	General Binary Tree
Search	O(log n)	O(n)	O(n)
Insert	O(log n)	O(n)	O(n)
Delete	O(log n)	O(n)	O(n)

Operation	BST (Balanced)	BST (Skewed)	General Binary Tree
Traversal	O(n)	O(n)	O(n)
Height	O(n)	O(n)	O(n)

Space Complexities

Operation	Space Complexity	Notes
Recursive Traversal	O(h)	Recursion stack
Iterative Traversal	O(w)	Queue for level order
Search (Recursive)	O(h)	Stack space
Search (Iterative)	O(1)	No extra space

Where:

- n = number of nodes
- h = height of tree
- w = maximum width of tree

Best Practices

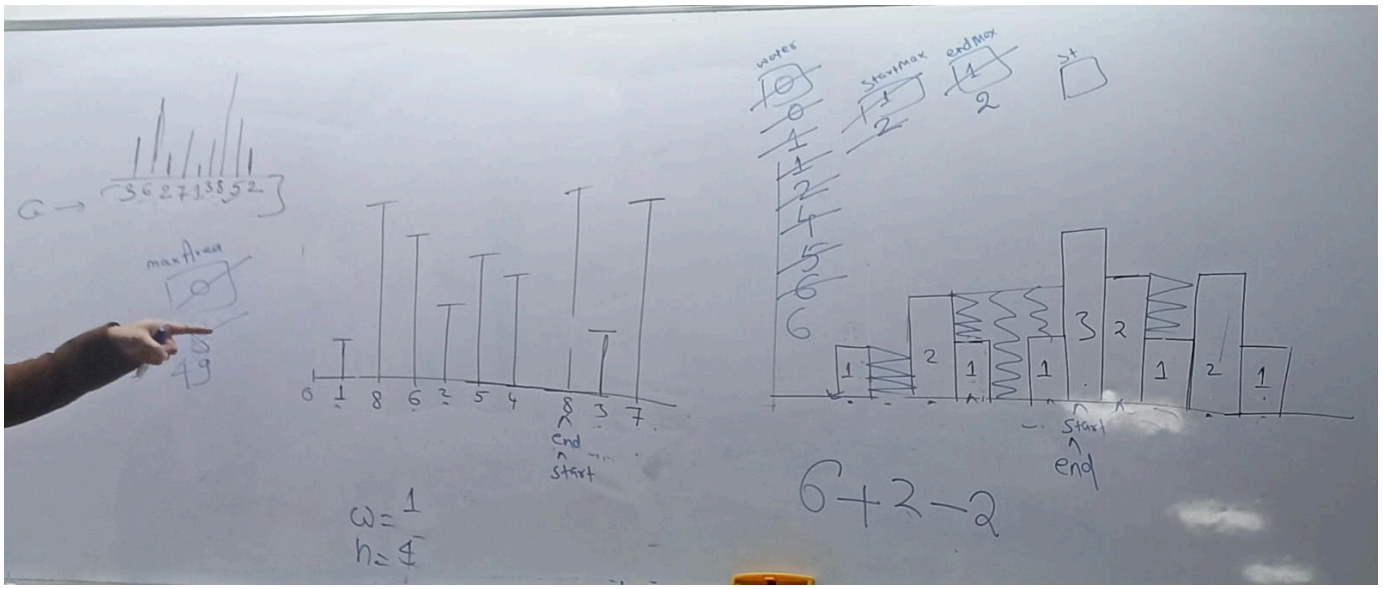
1. **Always check for null** before accessing node properties
2. **Use iterative approaches** when possible to save stack space
3. **For BST operations**, maintain the BST property
4. **Choose balanced trees** (AVL, Red-Black) for guaranteed O(log n) operations
5. **Use level order traversal** for problems requiring level-wise processing
6. **Consider space-time tradeoffs** when choosing recursive vs iterative

---

Common Patterns

1. **Two Pointer**: Left and right child pointers
2. **Recursion**: Most tree problems solved recursively
3. **DFS**: Preorder, Inorder, Postorder
4. **BFS**: Level order traversal using queue
5. **Backtracking**: Path finding problems
6. **Divide and Conquer**: Split problem into left and right subtrees

---



Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:46 AM

leetcode.com/problems/search-insert-position/

Problem List Submit Premium

Description | Editorial | Solutions | Submissions

35. Search Insert Position

Solved

Easy Topics Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Constraints:

18.3K 397 97 Online

Code

Java Auto

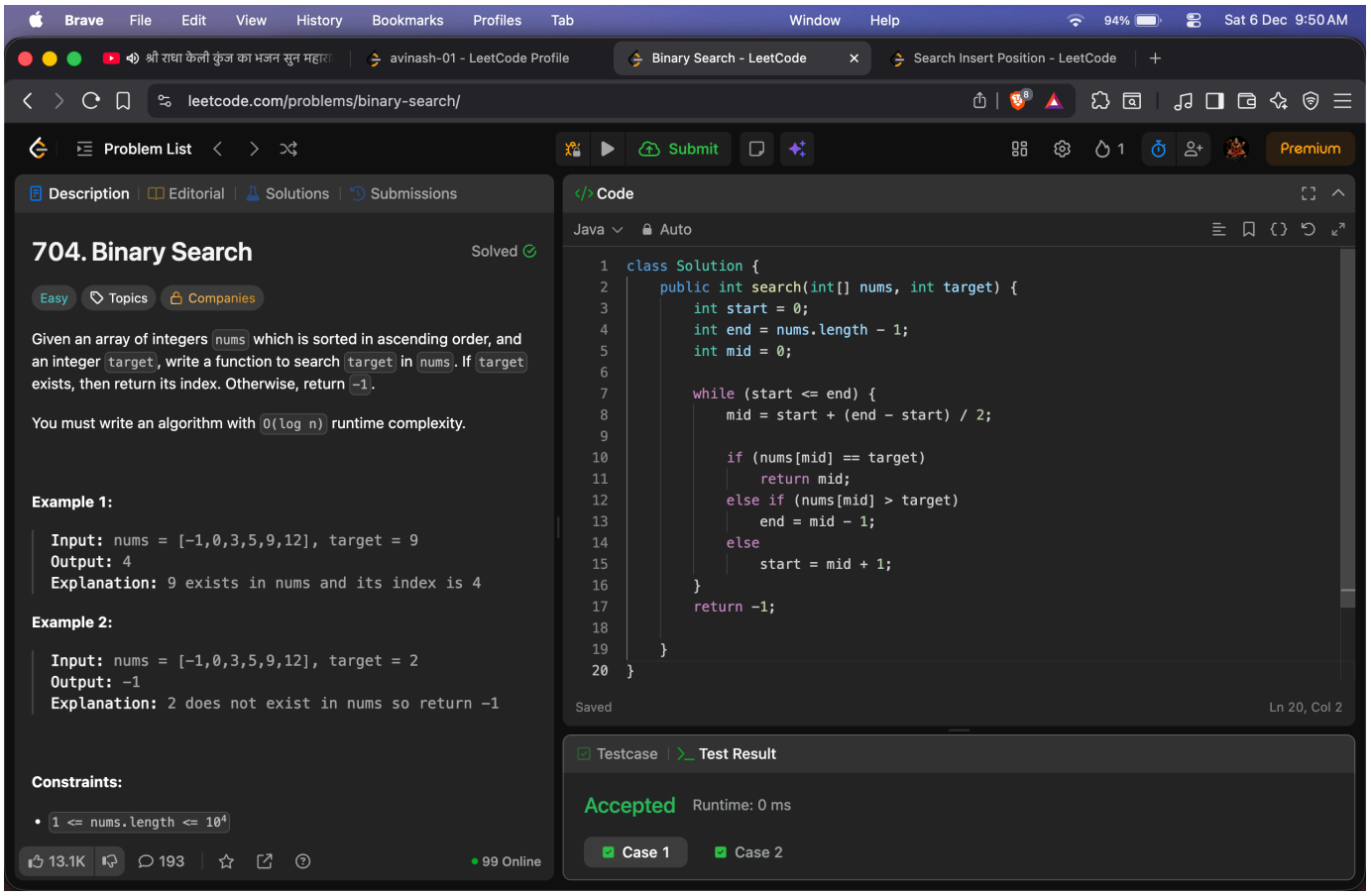
```java
class Solution {
    public int searchInsert(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;

        while (start <= end) {
            int mid = start + (end - start) / 2;

            if (nums[mid] == target)
                return mid;
            else if (nums[mid] > target)
                end = mid - 1;
            else
                start = mid + 1;
        }
        return start;
    }
}
```

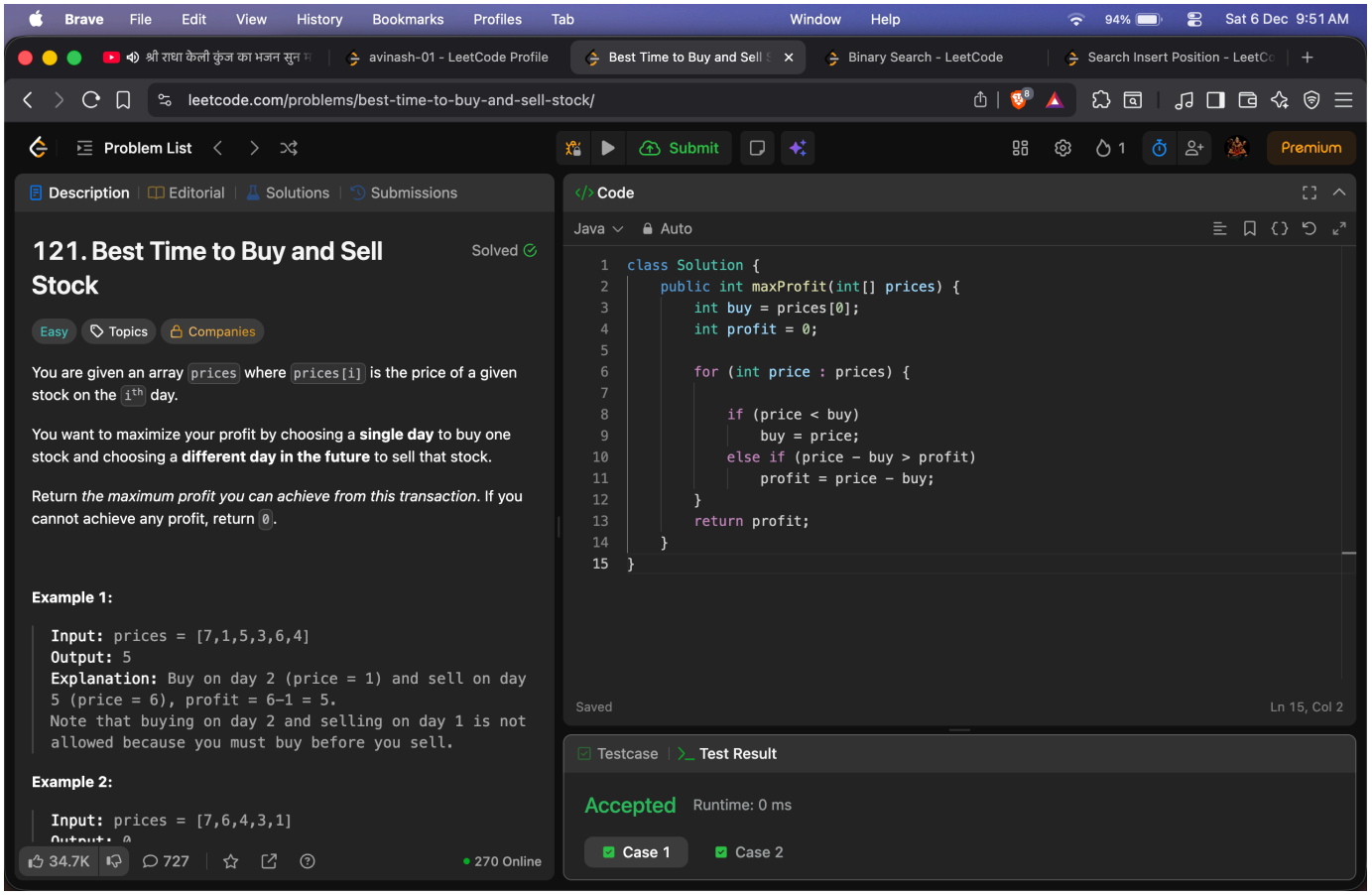
Saved Ln 18, Col 2

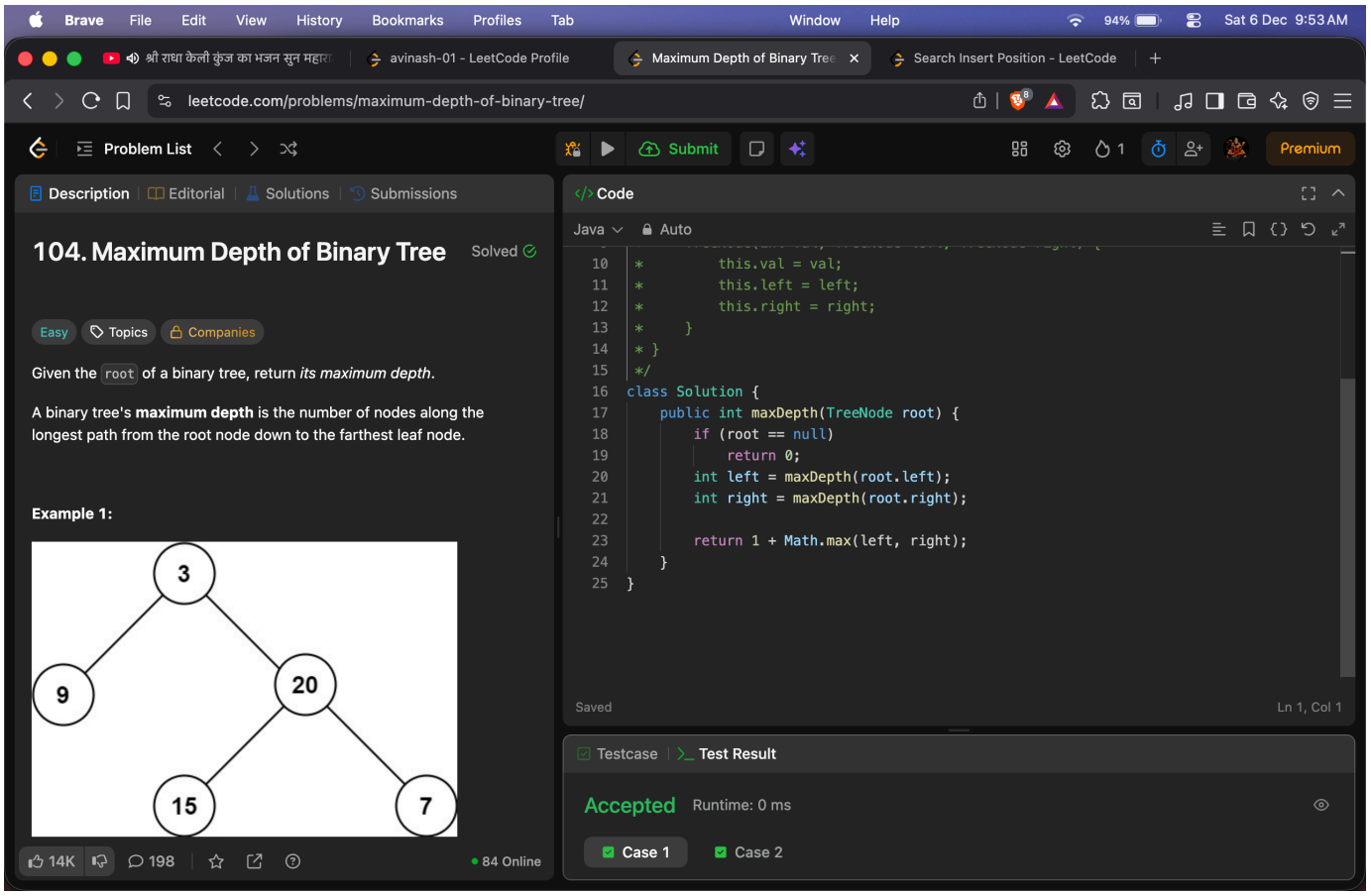
Testcase | Test Result

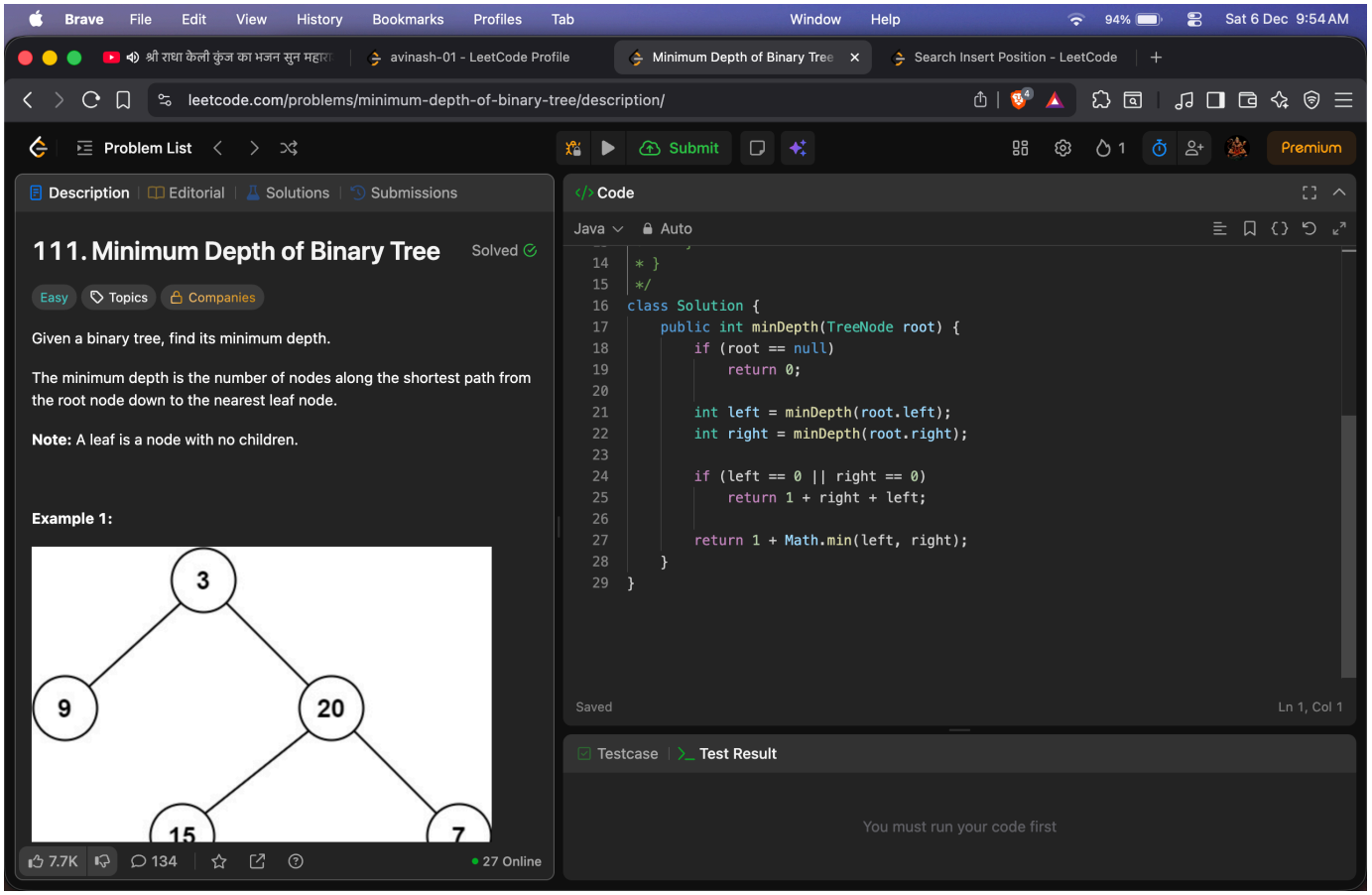
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3


Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:50 AM
avinash-01 - LeetCode Profile
Binary Search - LeetCode
Search Insert Position - LeetCode
leetcode.com/problems/binary-search/
Problem List
Submit
Premium
Description Editorial Solutions Submissions
704. Binary Search
Solved
Easy Topics Companies
Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.
You must write an algorithm with O(log n) runtime complexity.
Example 1:
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
Example 2:
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
Constraints:
1 <= nums.length <= 10^4
13.1K 193 99 Online
Code
Java Auto
class Solution {
public int search(int[] nums, int target) {
int start = 0;
int end = nums.length - 1;
int mid = 0;
while (start <= end) {
mid = start + (end - start) / 2;
if (nums[mid] == target)
return mid;
else if (nums[mid] > target)
end = mid - 1;
else
start = mid + 1;
}
return -1;
}
}
Saved
Ln 20, Col 2
Testcase Test Result
Accepted Runtime: 0 ms
Case 1 Case 2


Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:51 AM
avinash-01 - LeetCode Profile
Best Time to Buy and Sell
Binary Search - LeetCode
Search Insert Position - LeetCode
leetcode.com/problems/best-time-to-buy-and-sell-stock/
Problem List
Submit
Premium
Description Editorial Solutions Submissions
121. Best Time to Buy and Sell Stock
Solved
Easy Topics Companies
You are given an array prices where prices[i] is the price of a given stock on the ith day.
You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.
Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.
Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
Example 2:
Input: prices = [7,6,4,3,1]
34.7K 727 270 Online
Code
Java Auto
class Solution {
public int maxProfit(int[] prices) {
int buy = prices[0];
int profit = 0;
for (int price : prices) {
if (price < buy)
buy = price;
else if (price - buy > profit)
profit = price - buy;
}
return profit;
}
}
Saved
Ln 15, Col 2
Testcase Test Result
Accepted Runtime: 0 ms
Case 1 Case 2


Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:53 AM
avinash-01 - LeetCode Profile
Maximum Depth of Binary Tree
Search Insert Position - LeetCode
leetcode.com/problems/maximum-depth-of-binary-tree/
Problem List
Submit
Premium
Description
Editorial
Solutions
Submissions
104. Maximum Depth of Binary Tree
Solved
Easy
Topics
Companies
Given the root of a binary tree, return its maximum depth.
A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.
Example 1:
3
9
20
15
7
14K
198
84 Online
Code
Java
Auto
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public int maxDepth(TreeNode root) {
if (root == null)
return 0;
int left = maxDepth(root.left);
int right = maxDepth(root.right);
return 1 + Math.max(left, right);
}
}
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2


Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:54 AM
avinash-01 - LeetCode Profile
Minimum Depth of Binary Tree
Search Insert Position - LeetCode
leetcode.com/problems/minimum-depth-of-binary-tree/description/
Problem List
Submit
Premium
Description
Editorial
Solutions
Submissions
111. Minimum Depth of Binary Tree
Solved
Easy
Topics
Companies
Given a binary tree, find its minimum depth.
The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.
Note: A leaf is a node with no children.
Example 1:
3
9
20
15
7
7.7K
134
27 Online
Code
Java
Auto
* }
*/
class Solution {
public int minDepth(TreeNode root) {
if (root == null)
return 0;
int left = minDepth(root.left);
int right = minDepth(root.right);
if (left == 0 || right == 0)
return 1 + right + left;
return 1 + Math.min(left, right);
}
}
Saved
Ln 1, Col 1
Testcase
Test Result
You must run your code first

Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:55 AM

श्री राधा केली कुंज का भजन सुन महारा | avinash-01 - LeetCode Profile | Container With Most Water - L | Search Insert Position - LeetCode

leetcode.com/problems/container-with-most-water/

Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

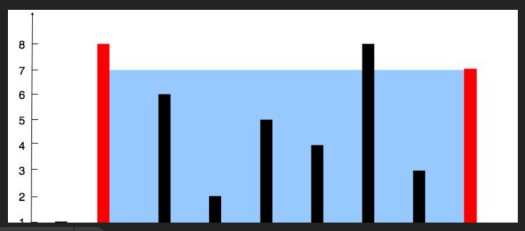
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



33.1K | 799 | 241 Online

Code

Java | Auto

```java
class Solution {
    public int maxArea(int[] height) {
        int start = 0;
        int end = height.length - 1;
        int maxArea = 0;

        while (start < end) {
            int width = end - start;
            int h = height[start] <= height[end] ? height[start++] : height[end--];

            int area = h * width;

            if (area > maxArea)
                maxArea = area;
        }
        return maxArea;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Brave File Edit View History Bookmarks Profiles Tab Window Help 94% Sat 6 Dec 9:56 AM

The Great Mantra: Hare Krishna | avinash-01 - LeetCode Profile | Trapping Rain Water - LeetCod | Search Insert Position - LeetCode

leetcode.com/problems/trapping-rain-water/

Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

42. Trapping Rain Water

Solved

Hard | Topics | Companies

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:

```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```

Example 2:

35.5K | 509 | 248 Online

Code

Java | Auto

```java
class Solution {
    public int trap(int[] height) {
        int start = 0, end = height.length-1, water = 0;
        int startMax = height[0], endMax = height[end];

        while(start < end){
            if(height[start] <= height[end]){
                if(height[start] > startMax)
                    startMax = height[start];
                else
                    water = water + startMax - height[start];

                start++;
            }else{
                if(height[end] > endMax)
                    endMax = height[end];
                else
                    water = water + endMax - height[end];

                end--;
            }
        }
        return water;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result