# “Data Structures (IS33)”

A report submitted on Leetcode questions.

**In**

**Third Semester**

**By**


| USN | Name of the Student |
|---|---|
| 1MS23IS155 | Atharv Dixit |

**Under the guidance of**

**Shivananda S**

Assistant Professor

Dept. of ISE, RIT



**Ramaiah Institute of Technology**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING RAMAIAH INSTITUTE OF TECHNOLOGY**

**(AUTONOMOUS INSTITUTE AFFILIATED TO VTU)**

**M. S. RAMAIAH NAGAR, M. S. R. I. T. POST, BANGALORE – 560054**

# Contents

# STACK –

## Problem 3174 – Clear Digits (Easy)

You are given a string s.

Your task is to remove **all** digits by doing this operation repeatedly:

- Delete the *first* digit and the **closest non-digit** character to its *left*.

Return the resulting string after removing all digits.

**Example:**

**Input:** s = "cb34"

**Output:** ""

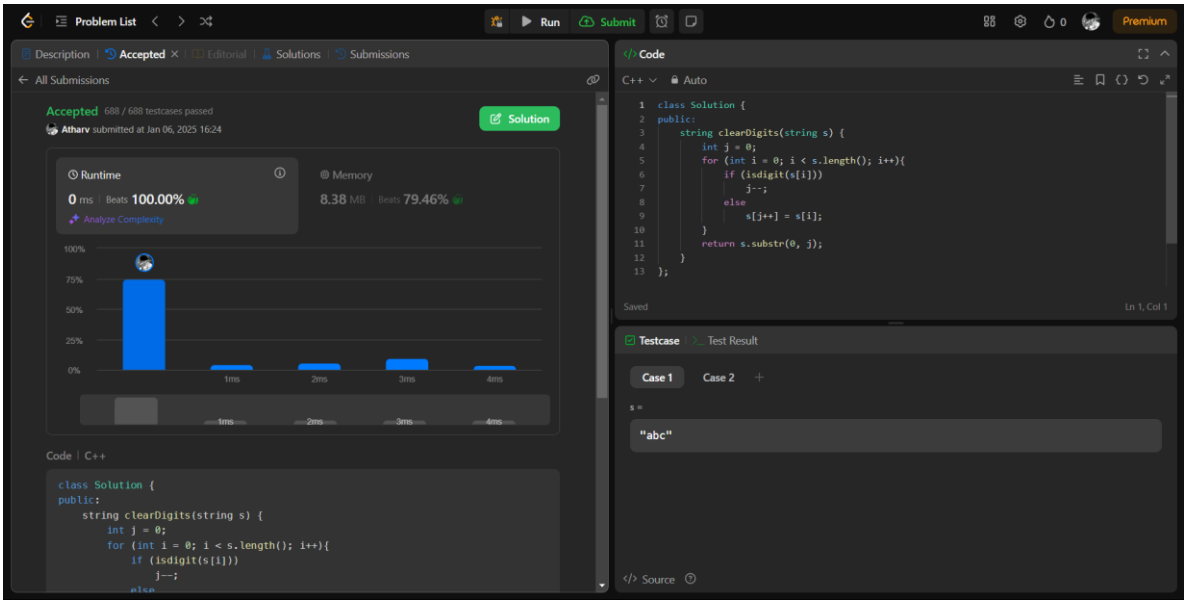
**Explanation:**

First, we apply the operation on s[2], and s becomes "c4".

Then we apply the operation on s[1], and s becomes "".

**Constraints:**

- 1 <= s.length <= 100
- s consists only of lowercase English letters and digits.
- The input is generated such that it is possible to delete all digits.



Problem 678 – Valid Parenthesis String (Medium)

Given a string s containing only three types of characters: '(', ')' and '*', return true *if* s *is* ***valid***.

The following rules define a **valid** string:

- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "".

**Example 1:**

**Input:** s = "(*)"

**Output:** true

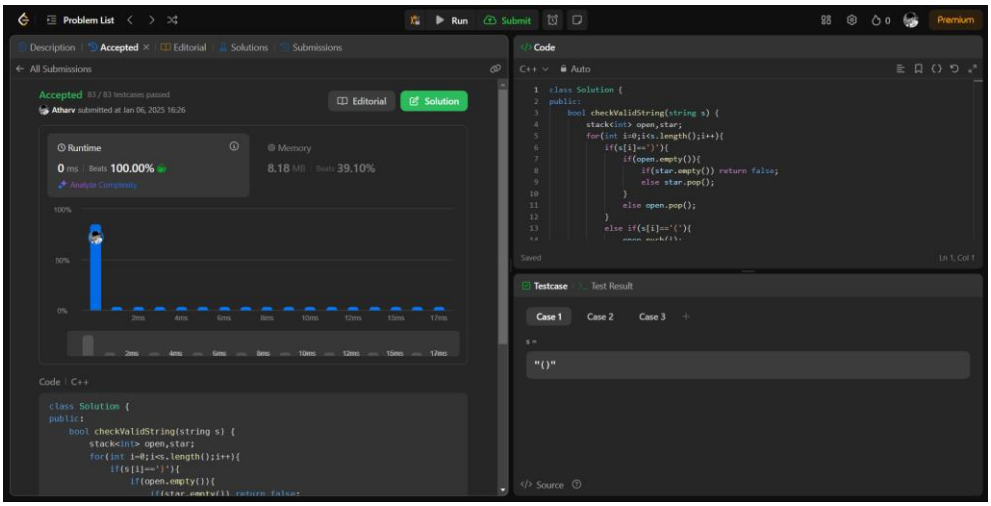
**Example 2:**

**Input:** s = "(*))"

**Output:** true

**Constraints:**

- 1 <= s.length <= 100
- s[i] is '(', ')' or '*'.



# QUEUE –

Problem 1700 – Number of Students Unable to Eat Lunch (Easy)

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a **stack**. At each step:

- If the student at the front of the queue **prefers** the sandwich on the top of the stack, they will **take it** and leave the queue.
- Otherwise, they will **leave it** and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer
arrays students and sandwiches where sandwiches[i] is the type of the i $^{th}$ sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the j$^{th}$ student in the initial queue (j = 0 is the front of the queue). Return *the number of students that are unable to eat.*

**Example 1:**

**Input:** students = [1,1,0,0], sandwiches = [0,1,0,1]

**Output:** 0

**Explanation:**

- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].

- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].

- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].

- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].

- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].

Hence all students are able to eat.

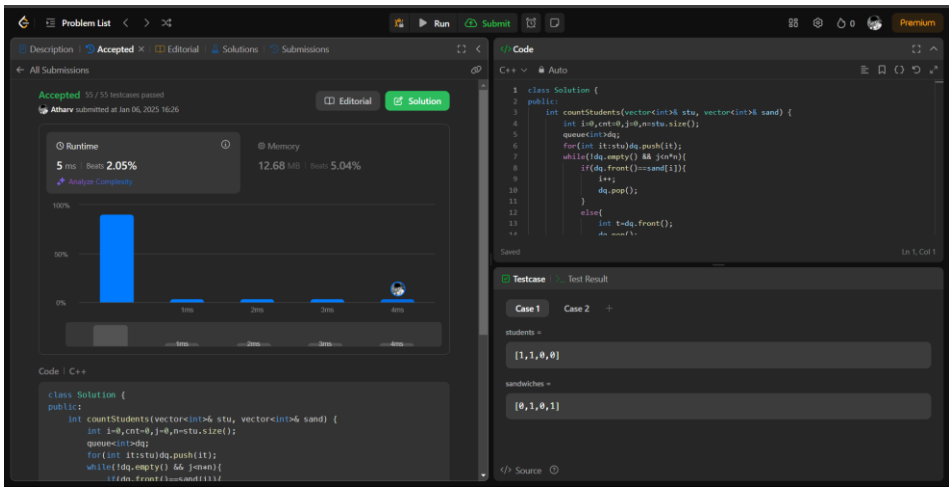
**Example 2:**

**Input:** students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]

**Output:** 3

**Constraints:**

- 1 <= students.length, sandwiches.length <= 100
- students.length == sandwiches.length
- sandwiches[i] is 0 or 1.
- students[i] is 0 or 1.



Problem 341- Flatten Nested List Iterator (Medium)

You are given a nested list of integers nestedList. Each element is either an integer or a list whose elements may also be integers or other lists. Implement an iterator to flatten it.

Implement the NestedIterator class:

- NestedIterator(List<NestedInteger> nestedList) Initializes the iterator with the nested list nestedList.
- int next() Returns the next integer in the nested list.
- boolean hasNext() Returns true if there are still some integers in the nested list and false otherwise.

Your code will be tested with the following pseudocode:

initialize iterator with nestedList

res = []

while iterator.hasNext()

  append iterator.next() to the end of res

return res

If res matches the expected flattened list, then your code will be judged as correct.

**Example 1:**

**Input:** nestedList = [[1,1],2,[1,1]]

**Output:** [1,1,2,1,1]

**Explanation:** By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,1,2,1,1].

**Example 2:**

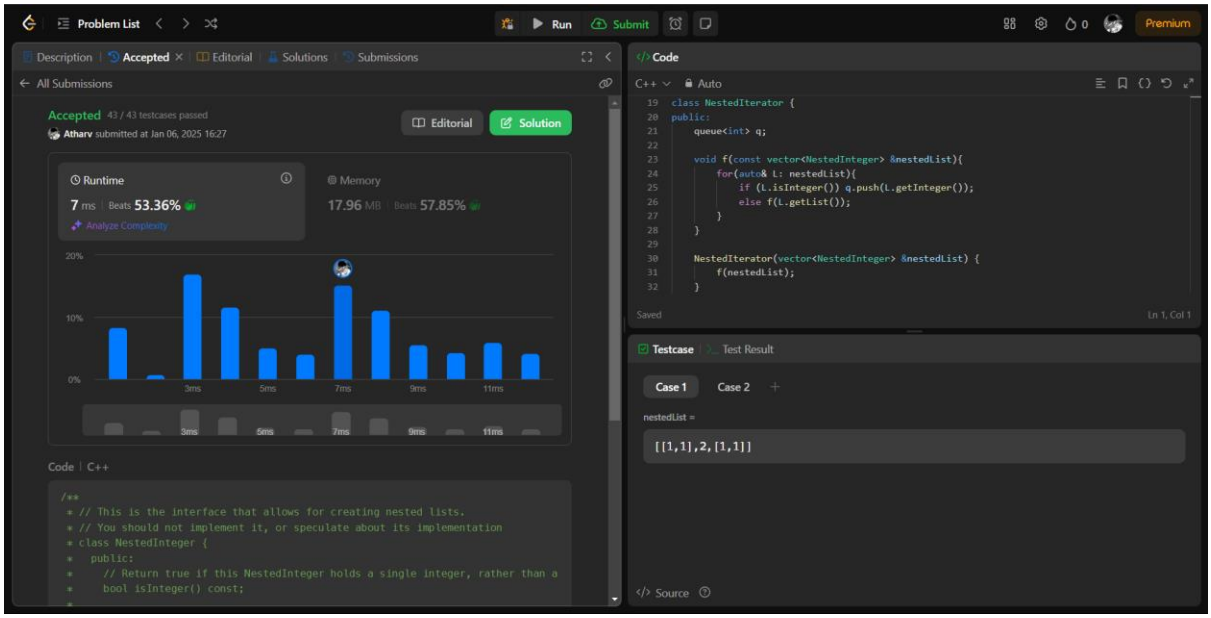
**Input:** nestedList = [1,[4,[6]]]

**Output:** [1,4,6]

**Explanation:** By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,4,6].

**Constraints:**

- 1 <= nestedList.length <= 500
- The values of the integers in the nested list is in the range [$-10^6$, $10^6$].



# LINKED LIST –

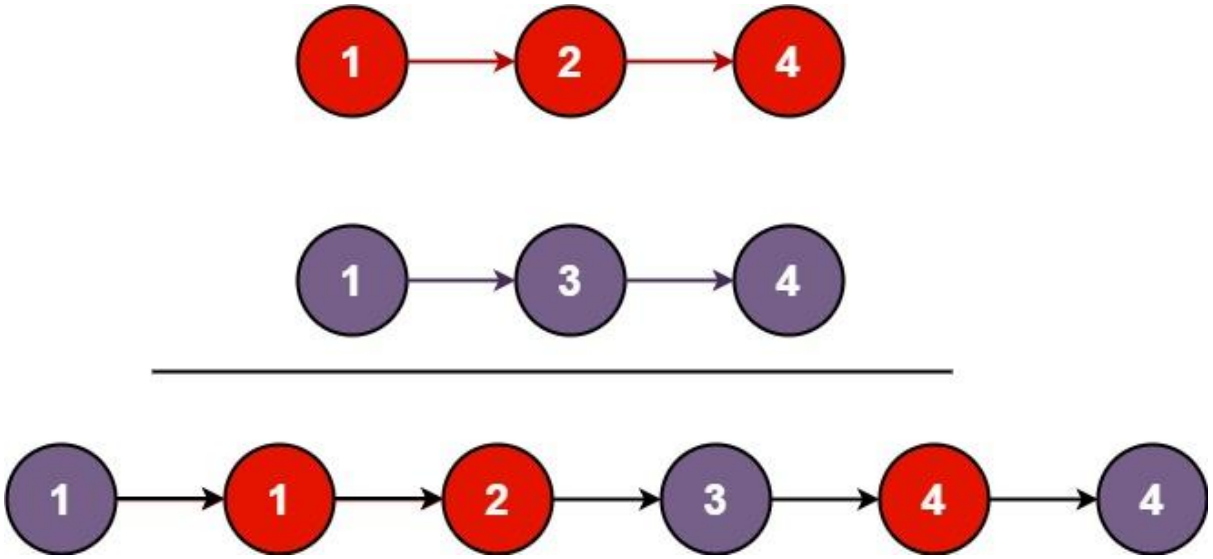
Problem 21– Merge Two Sorted Lists (Easy)

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

**Example 1:**




**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Example 2:**

**Input:** list1 = [], list2 = []

**Output:** []

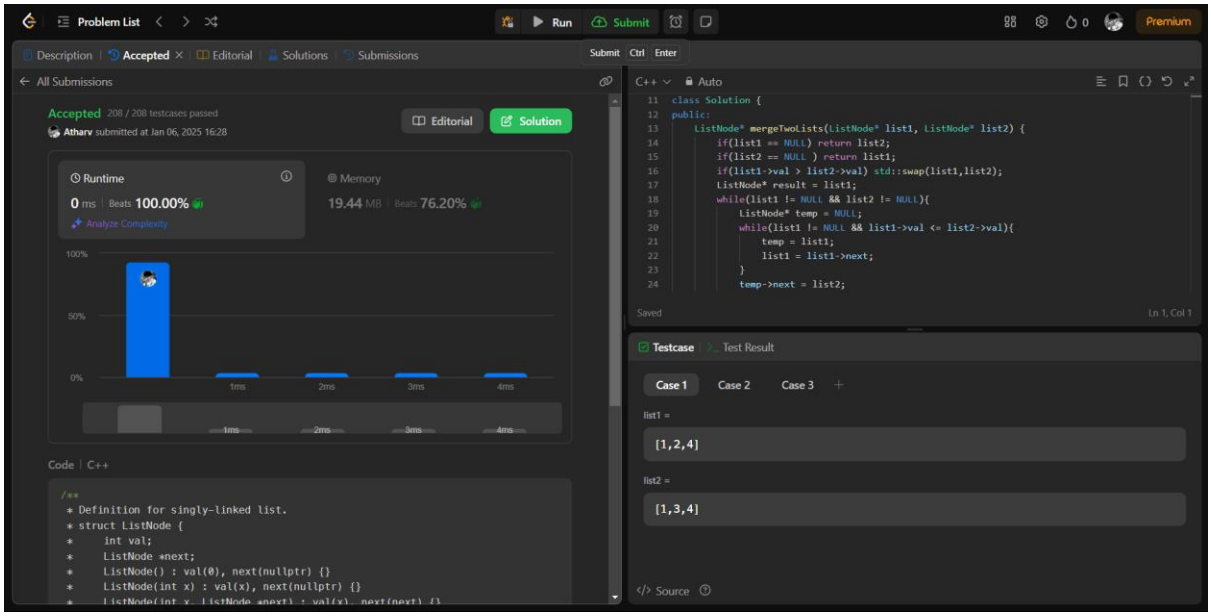
**Example 3:**

**Input:** list1 = [], list2 = [0]

**Output:** [0]

**Constraints:**

- The number of nodes in both lists is in the range [0, 50].
- -100 <= Node.val <= 100
- Both list1 and list2 are sorted in **non-decreasing** order.



Problem 706 – Design Hashmap (Easy)

Design a HashMap without using any built-in hash table libraries.

Implement the MyHashMap class:

- MyHashMap() initializes the object with an empty map.
- void put(int key, int value) inserts a (key, value) pair into the HashMap. If the key already exists in the map, update the corresponding value.
- int get(int key) returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key.
- void remove(key) removes the key and its corresponding value if the map contains the mapping for the key.

**Example 1:**

**Input**

["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]

[[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]

**Output**

[null, null, null, 1, -1, null, 1, null, -1]

**Explanation**

MyHashMap myHashMap = new MyHashMap();

myHashMap.put(1, 1); // The map is now [[1,1]]

myHashMap.put(2, 2); // The map is now [[1,1], [2,2]]

myHashMap.get(1);  // return 1, The map is now [[1,1], [2,2]]

myHashMap.get(3);  // return -1 (i.e., not found), The map is now [[1,1], [2,2]]

myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing value)

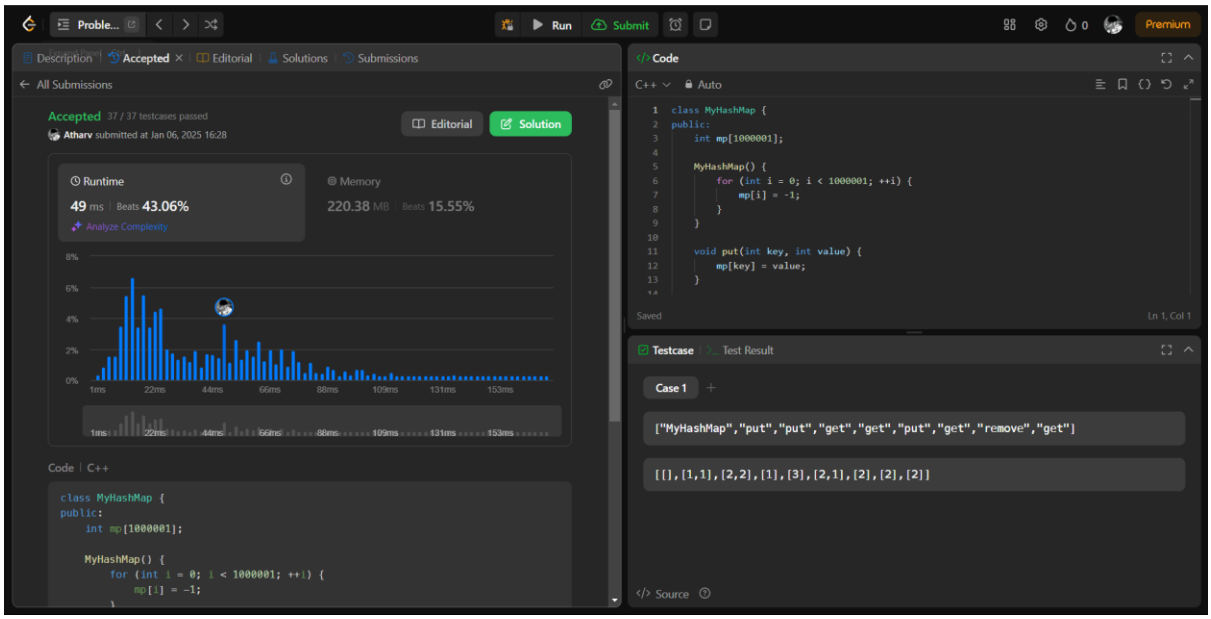
myHashMap.get(2);  // return 1, The map is now [[1,1], [2,1]]

myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]]

myHashMap.get(2);    // return -1 (i.e., not found), The map is now [[1,1]]

**Constraints:**

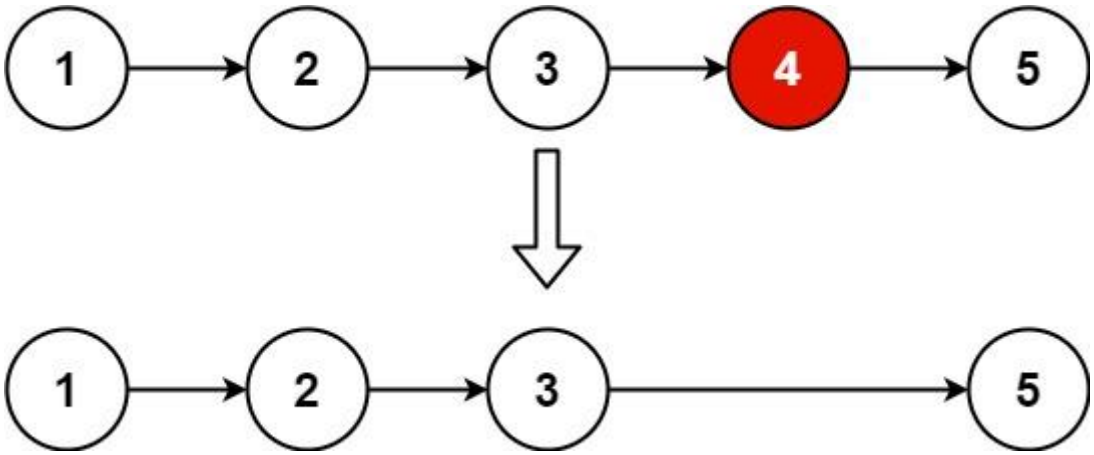
- 0 <= key, value <= $10^6$
- At most $10^4$ calls will be made to put, get, and remove.



Problem 19- Remove Nth Node from End of List (Medium)

Given the head of a linked list, remove the $n^{th}$ node from the end of the list and return its head.

**Example 1:**




**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

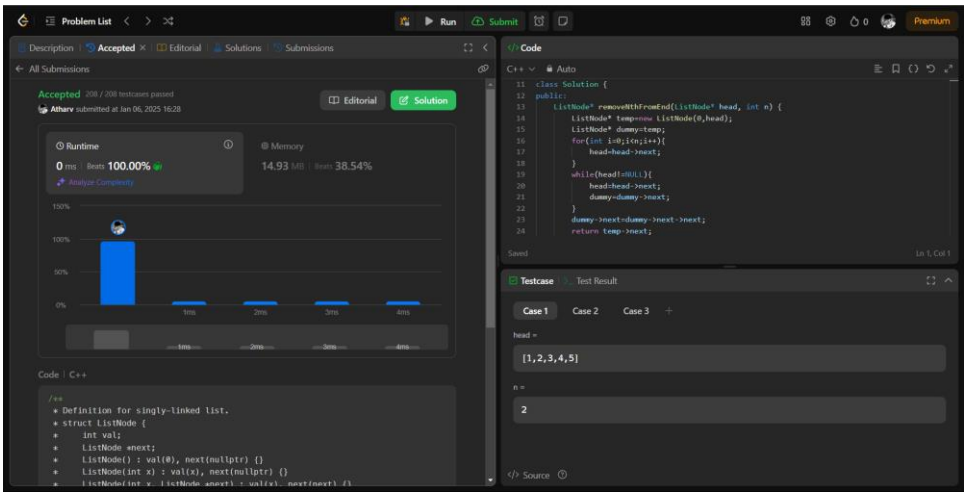
**Example 2:**

**Input:** head = [1], n = 1

**Output:** []

**Constraints:**

- The number of nodes in the list is sz.
- 1 <= sz <= 30
- 0 <= Node.val <= 100



## Problem 23- Merge k Sorted Lists (Hard)

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

**Example 1:**

**Input:** lists = [[1,4,5],[1,3,4],[2,6]]

**Output:** [1,1,2,3,4,4,5,6]

**Explanation:** The linked-lists are:

[

1->4->5,

1->3->4,

2->6

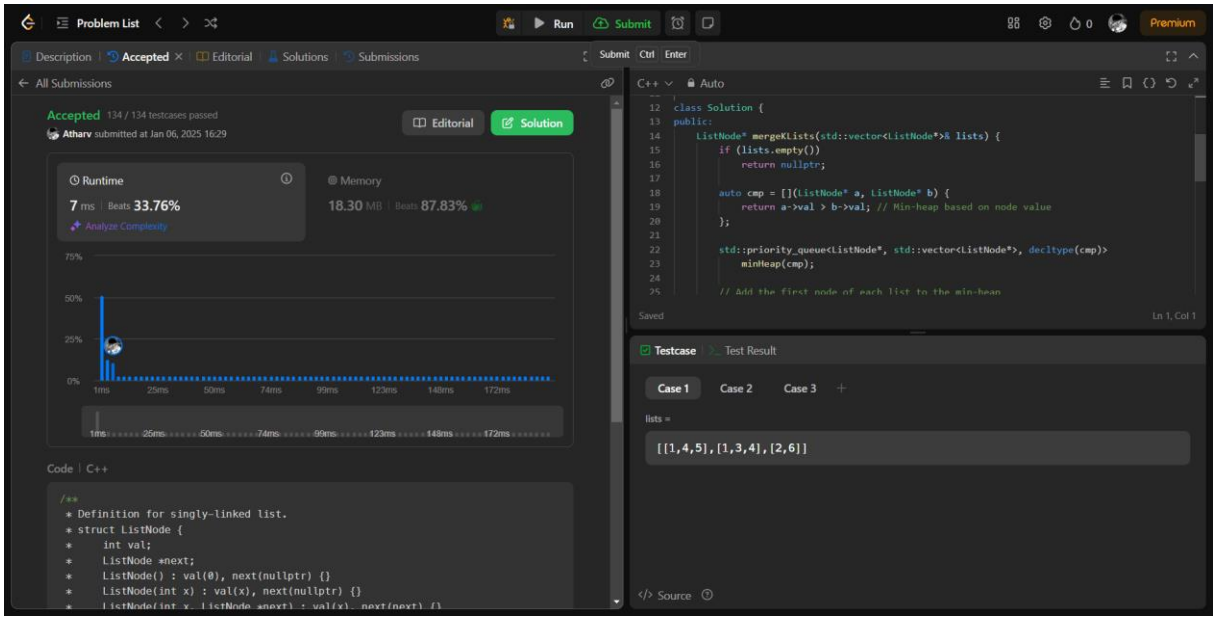
]

merging them into one sorted list:

1->1->2->3->4->4->5->6

**Constraints:**

- k == lists.length
- $0 <= k <= 10^4$
- 0 <= lists[i].length <= 500
- $-10^4 <= lists[i][j] <= 10^4$
- lists[i] is sorted in **ascending order**.



# TREES –

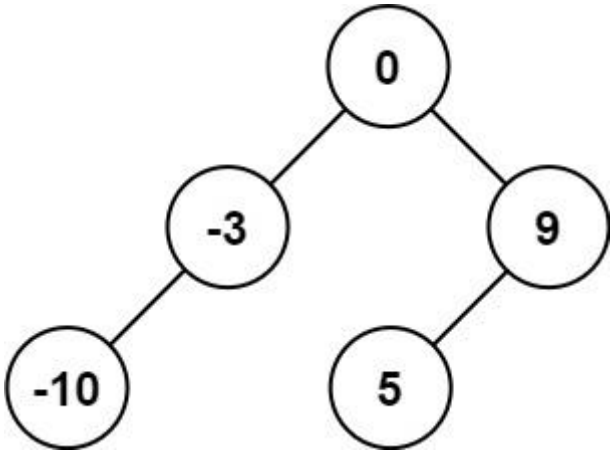
Problem 108– Covert Sorted Array to Binary Search Tree (Easy)

Given an integer array nums where the elements are sorted in **ascending order**, convert *it to a*

***height-balanced***

*binary search tree*.

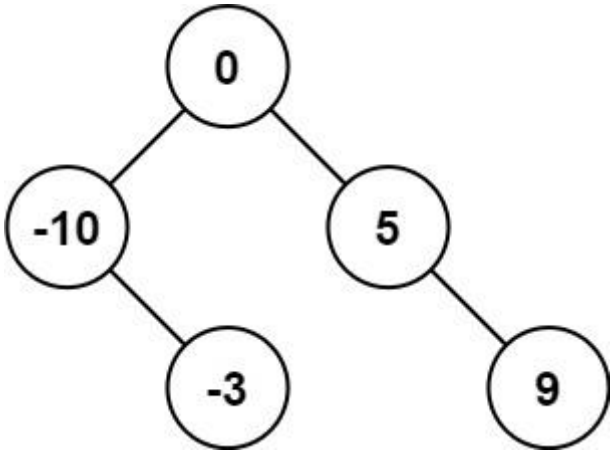
**Example 1:**



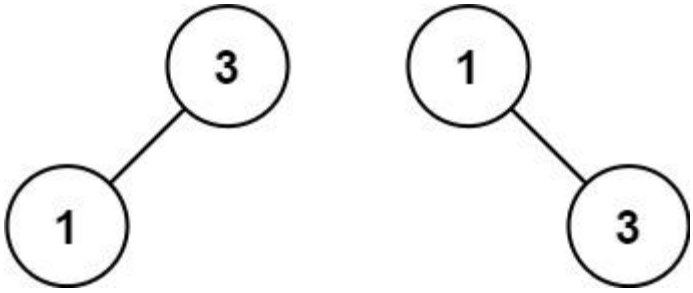

**Input:** nums = [-10,-3,0,5,9]

**Output:** [0,-3,9,-10,null,5]

**Explanation:** [0,-10,5,null,-3,null,9] is also accepted:




**Example 2:**



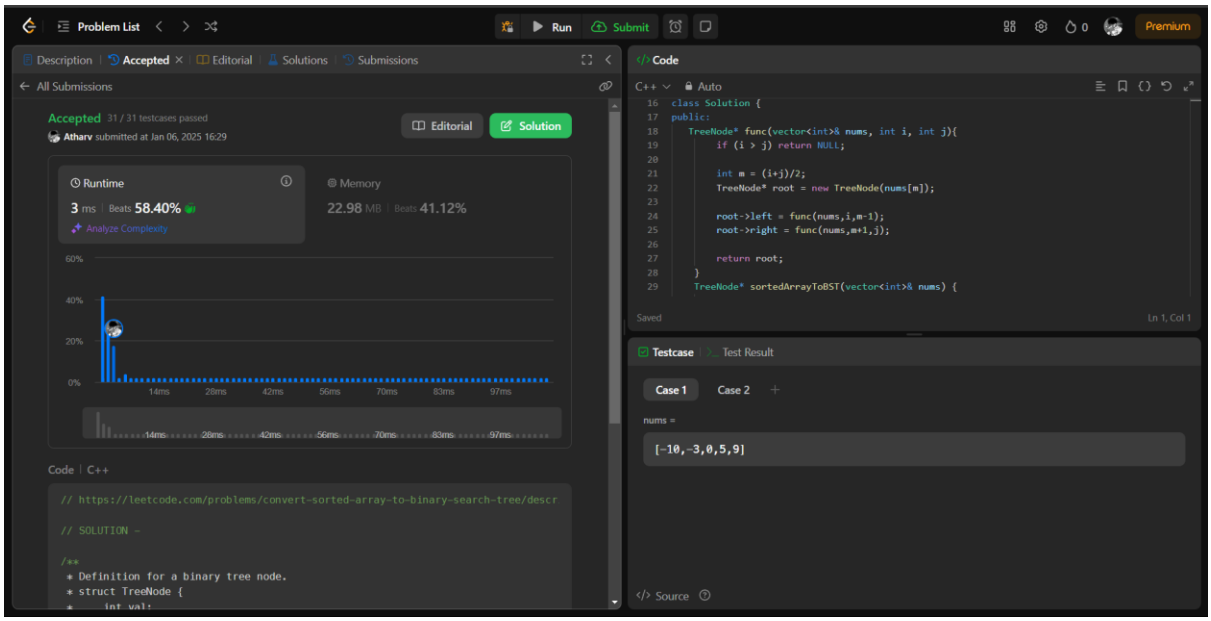

**Input:** nums = [1,3]

**Output:** [3,1]

**Explanation:** [1,null,3] and [3,1] are both height-balanced BSTs.

**Constraints:**

- 1 <= nums.length <= $10^4$
- $-10^4$ <= nums[i] <= $10^4$
- nums is sorted in a **strictly increasing** order.



Problem 94– Binary Tree Inorder Traversal (Easy)

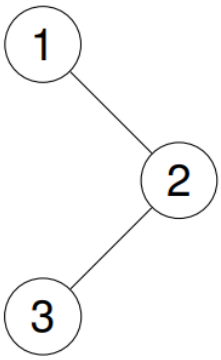
Given the root of a binary tree, return *the inorder traversal of its nodes' values*.

**Example 1:**

**Input:** root = [1,null,2,3]

**Output:** [1,3,2]

**Explanation:**

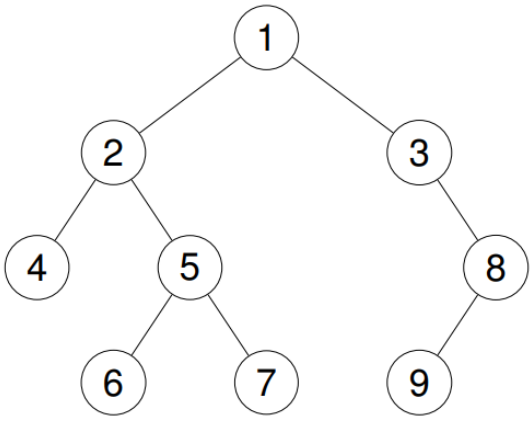



**Example 2:**

**Input:** root = [1,2,3,4,5,null,8,null,null,6,7,9]

**Output:** [4,2,6,5,7,1,3,9,8]

**Explanation:**



**Example 3:**

**Input:** root = []

**Output:** []

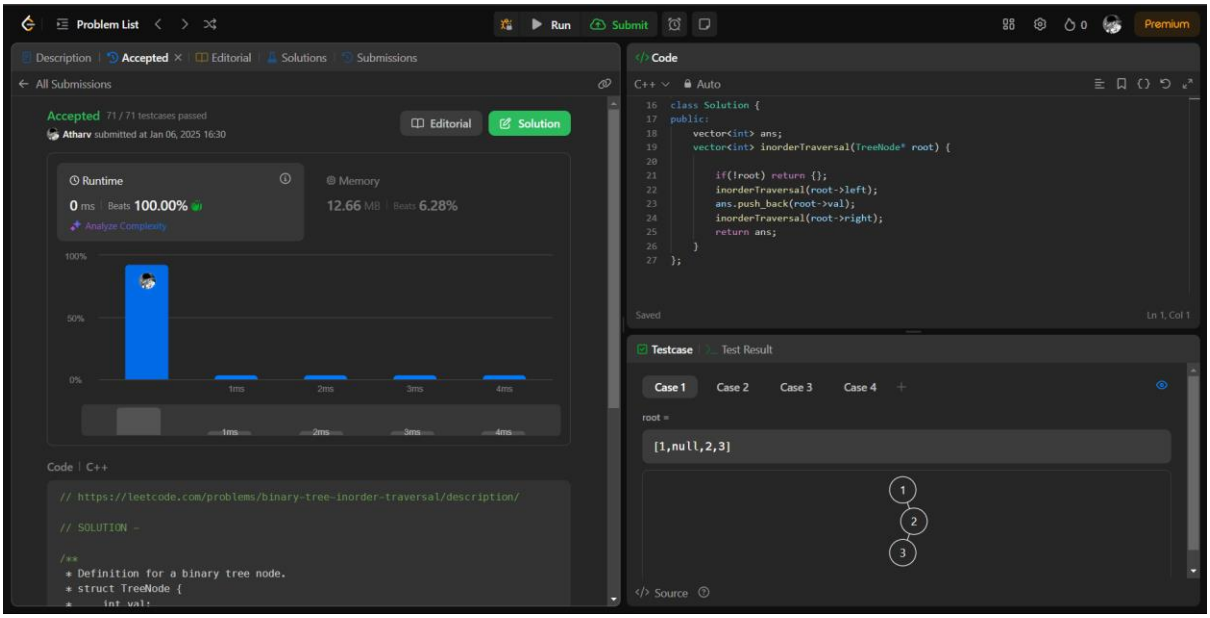
**Example 4:**

**Input:** root = [1]

**Output:** [1]

**Constraints:**

- The number of nodes in the tree is in the range [0, 100].
- -100 <= Node.val <= 100



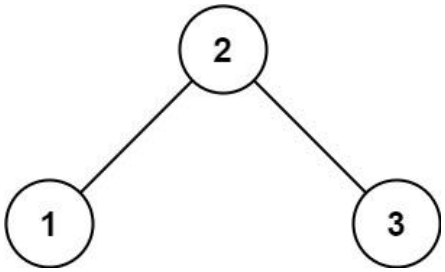
Problem 98- Validate Binary Search Tree (Medium)

Given the root of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

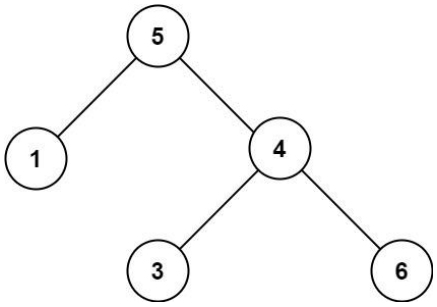
**Example 1:**




**Input:** root = [2,1,3]

**Output:** true

**Example 2:**




**Input:** root = [5,1,4,null,null,3,6]

**Output:** false

**Explanation:** The root node's value is 5 but its right child's value is 4.

**Constraints:**

- The number of nodes in the tree is in the range $[1, 10^4]$.
- $-2^{31} <= Node.val <= 2^{31} - 1$