

“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	3
Problem 3174 – Clear Digits (Easy)	3
Problem 678 – Valid Parenthesis String (Medium)	4
QUEUE	5
Problem 1700 – Number of Students Unable to Eat Lunch (Easy)	5
Problem 341 - Flatten Nested List Iterator (Medium)	7
LINKED LIST	9
Problem 21 – Merge Two Sorted Lists (Easy)	9
Problem 706 – Design Hashmap (Easy)	11
Problem 19 - Remove Nth Node from End of List (Medium)	13
Problem 23 - Merge k Sorted Lists (Hard)	14
TREES	15
Problem 108 – Covert Sorted Array to Binary Search Tree (Easy)	15
Problem 94 – Binary Tree Inorder Traversal (Easy)	17
Problem 98 - Validate Binary Search Tree (Medium)	19

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: $s = \text{"cb34"}$

Output: "c"

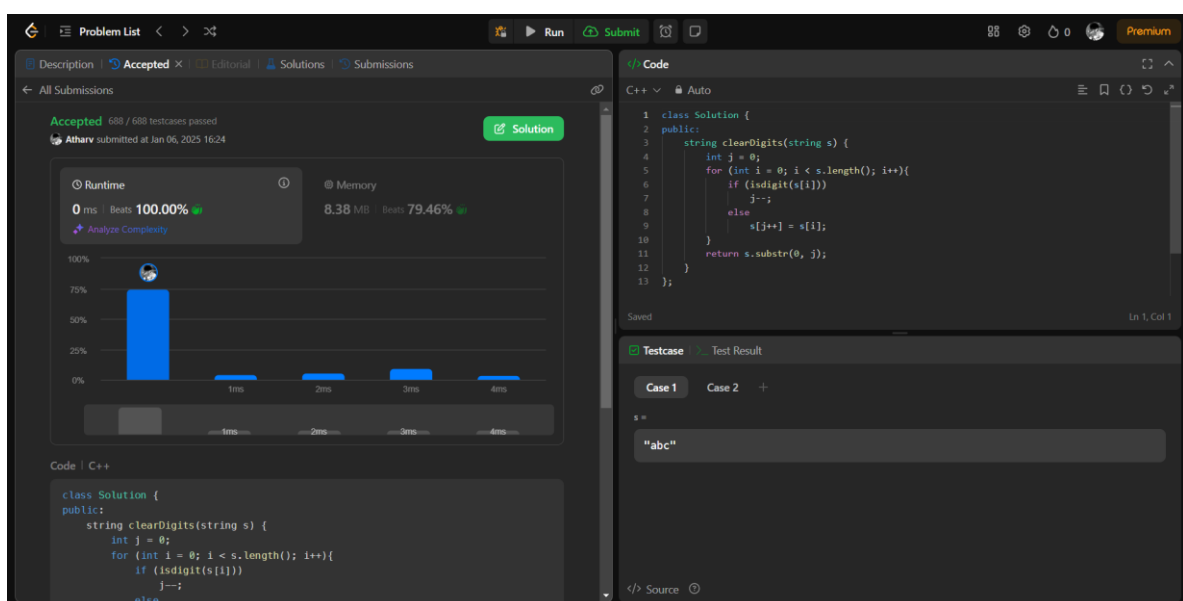
Explanation:

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

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

Constraints:

- $1 \leq s.length \leq 100$
- 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:

- $1 \leq s.length \leq 100$
- $s[i]$ is '(', ')' or '*'.

The screenshot displays the LeetCode submission interface for Problem 678. The left panel shows the submission status as 'Accepted' with 83/83 test cases passed. The runtime is 0 ms (beating 100.00%) and memory usage is 8.18 MB (beating 39.10%). A bar chart shows the performance of the submission compared to others. The right panel shows the C++ code for the solution, which uses a stack to validate the string. The test case input is '()' and the output is true.

```
class Solution {
public:
    bool checkValidString(string s) {
        stack<int> open_star;
        for(int i=0; i<s.length(); i++){
            if(s[i]=='('){
                if(open.empty()){
                    if(s[i]=='(') return false;
                } else star.pop();
            }
            else if(s[i]==')'){
                if(open.empty()) return false;
            }
        }
    }
};
```

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:

- $1 \leq \text{students.length}, \text{sandwiches.length} \leq 100$
- $\text{students.length} == \text{sandwiches.length}$
- $\text{sandwiches}[i]$ is 0 or 1.
- $\text{students}[i]$ is 0 or 1.

The screenshot displays a coding environment with the following components:

- Problem List:** Shows the current problem and navigation options.
- Description:** Contains the problem statement and constraints.
- Accepted:** Indicates the solution has passed all test cases (55/55).
- Runtime:** 5 ms, Beats 2.05%.
- Memory:** 12.68 MB, Beats 5.04%.
- Code:** A C++ solution using a queue to simulate the process.


```

class Solution {
public:
    int countStudents(vector<int>& stu, vector<int>& sand) {
        int i=0, cnt=0, j=0, n=stu.size();
        queue<int> dq;
        for(int i=0; i<n; i++) dq.push(stu[i]);
        while(!dq.empty() && j<n){
            if(dq.front() == sand[j]){
                dq.pop();
                j++;
            }
            else{
                int t=dq.front();
                dq.push(t);
            }
        }
        return cnt;
    }
};
      
```
- Testcase:** Case 1: students = [1,1,0,0], sandwiches = [0,1,0,1].

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:

- $1 \leq \text{nestedList.length} \leq 500$
- The values of the integers in the nested list is in the range $[-10^6, 10^6]$.

The screenshot displays a LeetCode submission interface for a C++ solution. The top navigation bar includes links for Problem List, Accepted, Editorial, Solutions, and Submissions. The submission status is "Accepted" with 43/43 testcases passed, submitted by Atharv on Jan 06, 2025 at 16:27. The performance metrics show a runtime of 7 ms (Beats 53.36%) and memory usage of 17.96 MB (Beats 57.85%). A bar chart visualizes the runtime performance across different time intervals. The C++ code defines a `NestedInteger` interface and a `NestedIterator` class. The test case input is `[[1,1],2,[1,1]]`.

Runtime Performance:

Time Interval	Runtime (ms)	Beats (%)
0-1ms	7	53.36
1-2ms	7	53.36
2-3ms	7	53.36
3-4ms	7	53.36
4-5ms	7	53.36
5-6ms	7	53.36
6-7ms	7	53.36
7-8ms	7	53.36
8-9ms	7	53.36
9-10ms	7	53.36
10-11ms	7	53.36
11-12ms	7	53.36

C++ Code:

```
19 class NestedInteger {
20 public:
21     queue<int> q;
22
23     void f(const vector<NestedInteger> &nestedList){
24         for(auto& l: nestedList){
25             if (l.isInteger()) q.push(l.getInteger());
26             else f(l.getList());
27         }
28     }
29
30     NestedIterator(vector<NestedInteger> &nestedList) {
31         f(nestedList);
32     }
33 }
```

Testcase:

Case 1 Case 2 +

nestedList =

```
[[1,1],2,[1,1]]
```


LINKED LIST –

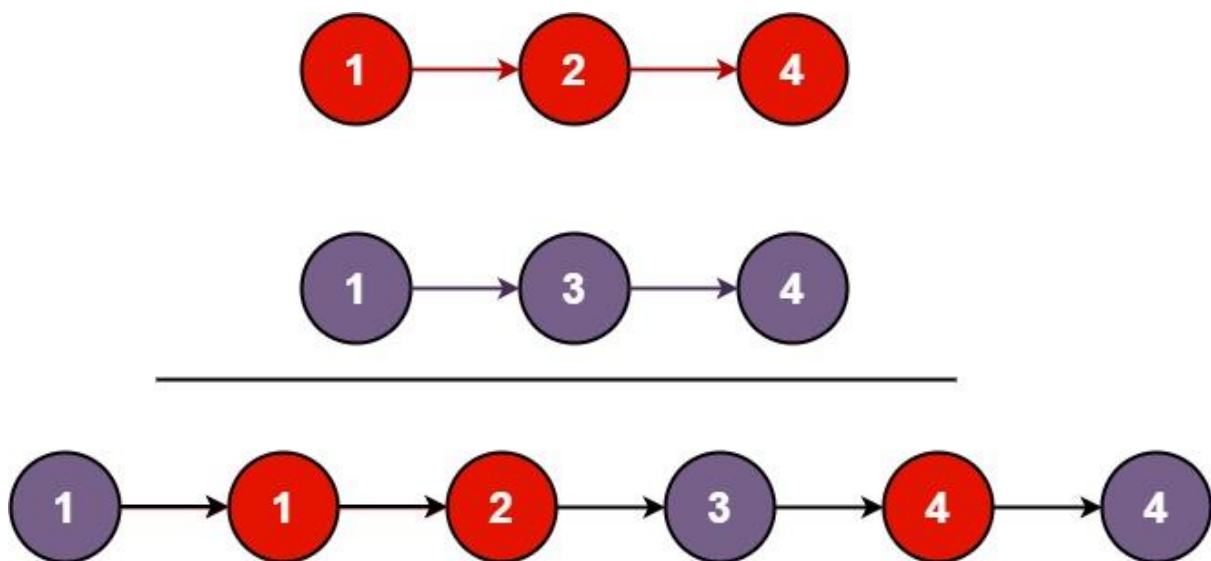
Problem 21– Merge Two Sorted Lists (Easy)

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].
- $-100 \leq \text{Node.val} \leq 100$
- Both list1 and list2 are sorted in **non-decreasing** order.

The screenshot shows a coding platform interface with a C++ solution for merging two sorted linked lists. The solution is accepted, with a runtime of 0 ms and memory usage of 19.44 MB. The code defines a `ListNode` struct and a `mergeTwoLists` function that merges two sorted linked lists into a new sorted linked list.

Runtime: 0 ms | Beats 100.00%
Memory: 19.44 MB | Beats 76.20%

Code (C++):

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1 == NULL) return list2;
        if(list2 == NULL) return list1;
        if(list1->val > list2->val) std::swap(list1, list2);
        ListNode* result = list1;
        while(list1 != NULL && list2 != NULL){
            ListNode* temp = NULL;
            while(list1 != NULL && list1->val <= list2->val){
                temp = list1;
                list1 = list1->next;
            }
            temp->next = list2;
        }
        return result;
    }
};
```

Testcase:

Case 1: list1 = [1,2,4], list2 = [1,3,4]

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)
```

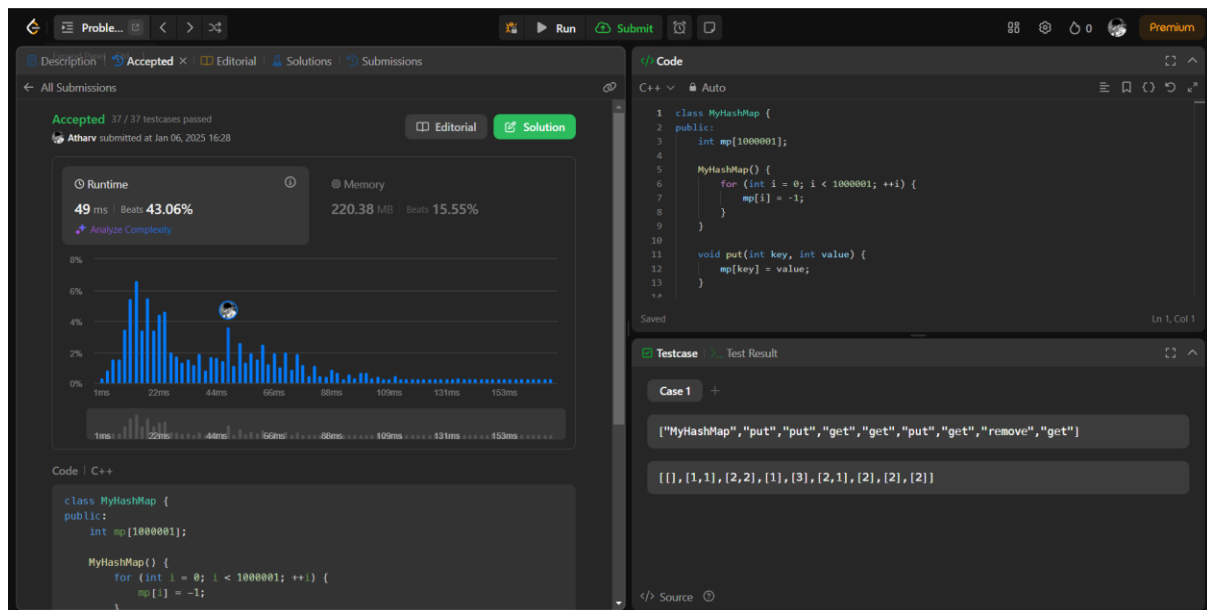
```
myHashMap.get(2); // return 1, The map is now [[1,1], [2,1]]
```

```
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:

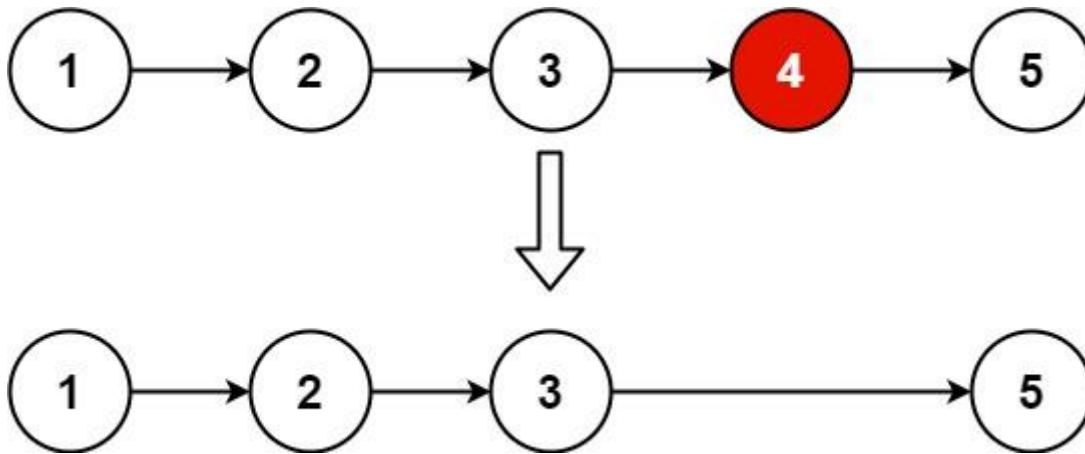
- $0 \leq \text{key}, \text{value} \leq 10^6$
- At most 10^4 calls will be made to put, get, and remove.



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.
- $1 \leq sz \leq 30$
- $0 \leq \text{Node.val} \leq 100$

Accepted 208 / 208 testcases passed
Atharv submitted at Jan 06, 2023 16:28

Runtime 0 ms | **Beats** 100.00% | **Memory** 14.93 MB | **Beats** 38.54%

Code

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* temp = head;
        for(int i=0; i<n; i++){
            head = head->next;
        }
        while(head->next){
            head = head->next;
            temp = temp->next;
        }
        temp->next = temp->next->next;
        return head;
    }
};
```

Testcase | **Test Result**

Case 1 | Case 2 | Case 3

head = [1,2,3,4,5]

n = 2

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:

- $k == \text{lists.length}$
- $0 \leq k \leq 10^4$
- $0 \leq \text{lists}[i].\text{length} \leq 500$
- $-10^4 \leq \text{lists}[i][j] \leq 10^4$
- $\text{lists}[i]$ is sorted in **ascending order**.

The screenshot shows a coding platform interface with a C++ solution for the 'Merge k Sorted Lists' problem. The solution is accepted, with a runtime of 7 ms (beats 33.76%) and memory usage of 18.30 MB (beats 87.83%). The code uses a min-heap to merge the k sorted linked lists. The test case input is lists = [[1,4,5],[1,3,4],[2,6]] and the output is [1,1,2,3,4,4,5,6].

```
class Solution {
public:
    ListNode* mergeKLists(std::vector<ListNode*>& lists) {
        if (lists.empty())
            return nullptr;
        auto cmp = [](ListNode* a, ListNode* b) {
            return a->val > b->val; // Min-heap based on node value
        };
        std::priority_queue<ListNode*, std::vector<ListNode*>, decltype(cmp)>
minheap(cmp);
        // Add the first node of each list to the min-heap
    }
};
```

Definition for singly-linked list.

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};
```

TREES —

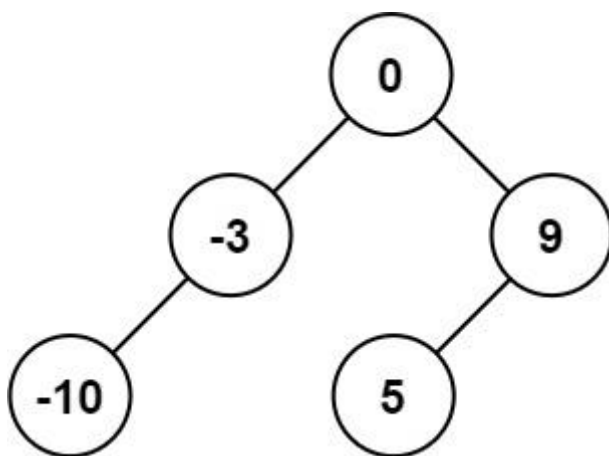
Problem 108— Covert Sorted Array to Binary Search Tree (Easy)

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

height-balanced

binary search tree.

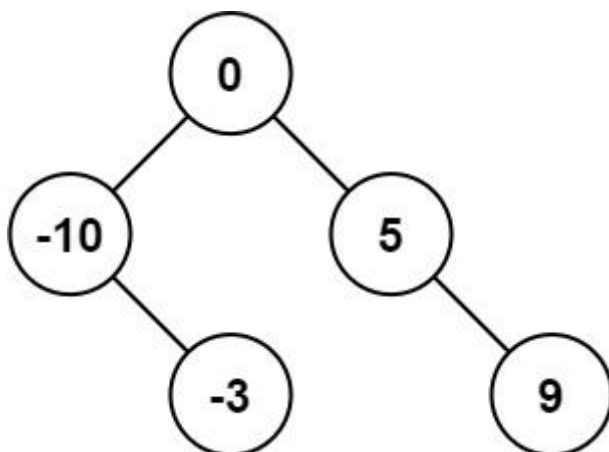
Example 1:



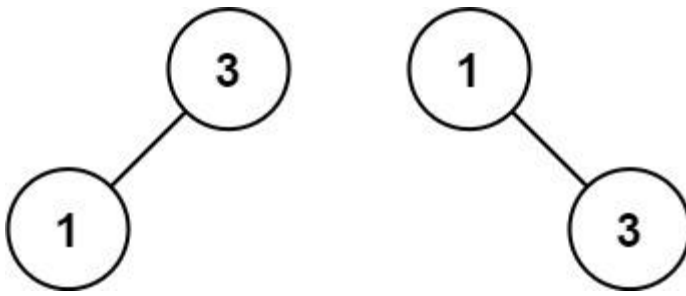
Input: `nums = [-10,-3,0,5,9]`

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:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- nums is sorted in a **strictly increasing** order.

The screenshot shows a LeetCode problem page for "Convert Sorted Array to Binary Search Tree". The solution is accepted, with a runtime of 3 ms (beats 58.40%) and memory usage of 22.98 MB (beats 41.12%). The code is written in C++ and implements a recursive function to build a height-balanced BST from a sorted array.

Code:

```

class Solution {
public:
    TreeNode* func(vector<int>& nums, int i, int j){
        if (i > j) return NULL;
        int m = (i+j)/2;
        TreeNode* root = new TreeNode(nums[m]);
        root->left = func(nums,i,m-1);
        root->right = func(nums,m+1,j);
        return root;
    }
    TreeNode* sortedArrayToBST(vector<int>& nums) {
  
```

Testcase:

Case 1: nums = [-10,-3,0,5,9]

Problem 94– Binary Tree Inorder Traversal (Easy)

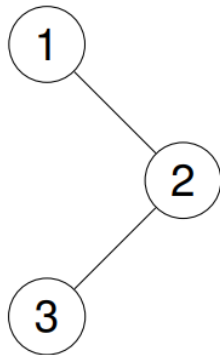
Given the root of a binary tree, return *the inorder traversal of its nodes' values*.

Example 1:

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

Output: [1,3,2]

Explanation:

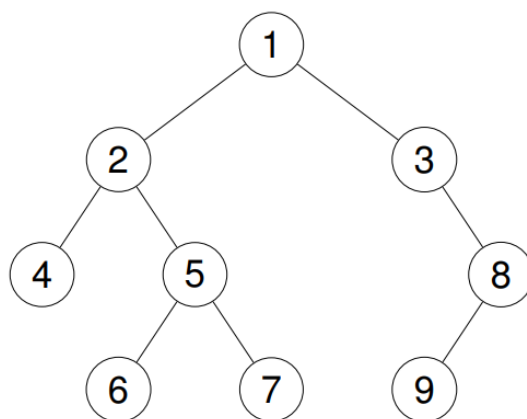


Example 2:

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].
- $-100 \leq \text{Node.val} \leq 100$

The screenshot displays the LeetCode submission page for the problem "Binary Tree Inorder Traversal". The top navigation bar includes "Problem List", "Run", "Submit", and "Premium". The main content area is divided into several sections:

- Description:** Shows the problem title and a link to the description.
- Accepted:** Indicates that 71/71 test cases passed. The submission was made by "Atharv" on Jan 06, 2025, at 16:30.
- Runtime and Memory:** The runtime is 0 ms, which is 100.00% faster than other submissions. The memory usage is 12.66 MB, which is 6.28% less than other submissions.
- Code:** A C++ solution is shown, implementing an inorder traversal using a recursive function. The code is as follows:

```
16 class Solution {
17 public:
18     vector<int> ans;
19     vector<int> inorderTraversal(TreeNode* root) {
20
21         if(!root) return {};
22         inorderTraversal(root->left);
23         ans.push_back(root->val);
24         inorderTraversal(root->right);
25         return ans;
26     }
27 }
```
- Testcase:** A test case is shown with the input "root = [1,null,2,3]". Below the input, a diagram of the resulting binary tree is displayed, showing a root node with a left child (1), a right child (2), and a right child (3) of node 2.

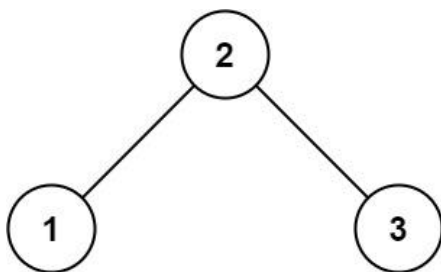
Problem 98- Validate Binary Search Tree (Medium)

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.

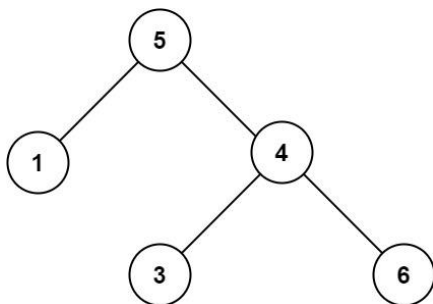
Example 1:



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 $[1, 10^4]$.
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

The screenshot displays a LeetCode submission interface for the problem "Validate Binary Search Tree". The submission status is "Accepted", indicating that all 86 test cases passed. The user, Atharv, submitted the solution on January 06, 2025, at 16:30. The performance metrics show a runtime of 0 ms (beating 100.00% of submissions) and a memory usage of 22.14 MB (beating 5.90% of submissions). The code is written in C++ and implements an inorder traversal to validate the BST. A test case [2,1,3] is shown, along with a diagram of the resulting binary tree structure.

Runtime: 0 ms | Beats 100.00%
Memory: 22.14 MB | Beats 5.90%

Code (C++):

```
19 class Solution {
20 public:
21     void inorder(TreeNode* root, vector<int> &ans){
22         if(root==NULL){
23             return;
24         }
25         inorder(root->left,ans);
26         ans.push_back(root->val);
27         inorder(root->right,ans);
28     }
29     bool isValidBST(TreeNode* root) {
30         vector<int>ans;
31         inorder(root,ans);
32     }
33 }
```

Testcase: Case 1: [2,1,3]

Tree Diagram:

```
graph TD
    2((2)) --> 1((1))
    2 --> 3((3))
```