VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

DATA STRUCTURES AAT REPORT
on

CODING CHALLENGES

Submitted by

GARVIT MUNDRA (1WN24CS102)

Under the Guidance of
Prof. Manjula S
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
August to December 2025

B.M.S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, GARVIT MUNDRA (1WN24CS102) student of 3rd Semester S Section, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that, this Coding Challenges for the course Data Structures (23CS3PCDST) has been carried out by me under the guidance of Prof. Prameetha Pai, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester August to December 2025.

I also declare that to the best of our knowledge and belief, the development reported here is not from part of any other report by any other students.

Signature

GARVIT MUNDRA (1WN24CS102)

Coding challenge 1:

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

- Leet code link :

https://leetcode.com/problems/min-stack/description/

Constraints:

- **$-2^{31} <= val <= 2^{31} - 1$**
- **Methods pop, top and getMin operations will always be called on non-empty stacks.**
- **At most $3 * 10^4$ calls will be made to push, pop, top, and getMin.**

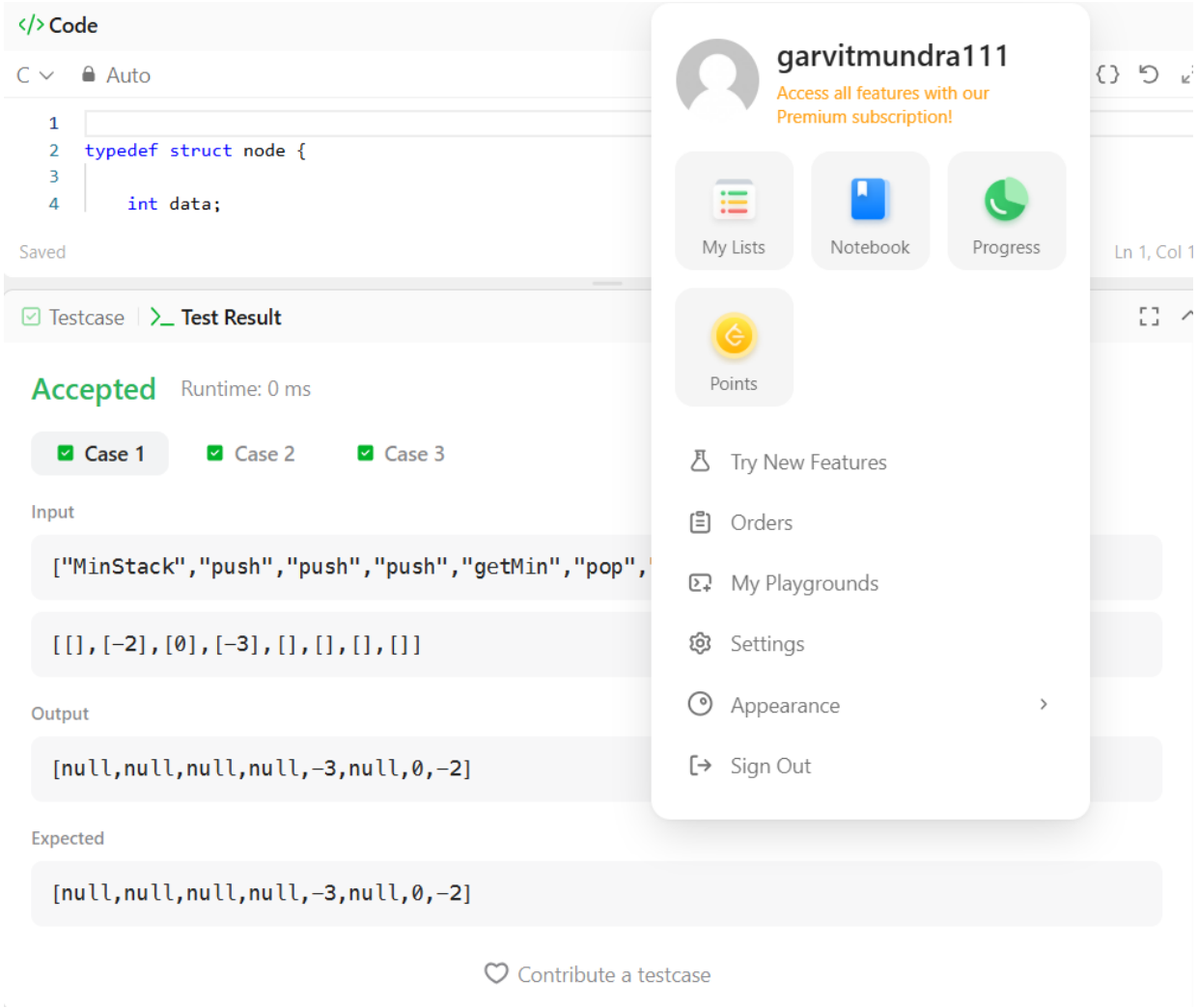
CODE:

```
typedef struct node {

    int data;
    struct node *next;
    struct node *prevMin;

} Node;

typedef struct {

    struct node *top1;
    struct node *top2;  /// to maintain min values in order

} MinStack;

MinStack* minStackCreate() {

    MinStack *stack = (MinStack *) malloc(sizeof(MinStack));

    stack->top1 = NULL;
    stack->top2 = NULL;

    return stack;

}

void minStackPush(MinStack* obj, int val) {

    Node *newNode = (Node *) malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = obj->top1;
    newNode->prevMin = NULL;
    obj->top1 = newNode;

    if(obj->top2 == NULL || obj->top2->data > val)
    {
        newNode->prevMin = obj->top2;
        obj->top2 = newNode;
```

```
    }

}

void minStackPop(MinStack* obj) {

    Node *deletedNode = obj->top1;

    if(obj->top1 == obj->top2)
        obj->top2 = obj->top2->prevMin;

    obj->top1 = obj->top1->next;
    free(deletedNode);

}

int minStackTop(MinStack* obj) {

    return obj->top1->data;

}

int minStackGetMin(MinStack* obj) {

    return obj->top2->data;

}

void minStackFree(MinStack* obj) {

    Node *deletedNode;

    while(obj->top1)
    {
        deletedNode = obj->top1;
        obj->top1 = obj->top1->next;
        free(deletedNode);
    }

    obj->top2 = NULL;
```

```
}

/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);

 * minStackPop(obj);

 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);

 * minStackFree(obj);
*/
```

TESTCASES:


Code
C
Auto
1
2 typedef struct node {
3
4 int data;
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
["MinStack","push","push","push","getMin","pop",
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Expected
[null,null,null,null,-3,null,0,-2]
Contribute a testcase
garvitmundra111
Access all features with our Premium subscription!
My Lists
Notebook
Progress
Points
Try New Features
Orders
My Playgrounds
Settings
Appearance
Sign Out

Coding Challenge 2:

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 linked list:
1->1->2->3->4->4->5->6
Example 2:
Input: lists = []
Output: []
Example 3:
Input: lists = [[]]
Output: []

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.**
- **The sum of lists[i].length will not exceed 10^4**

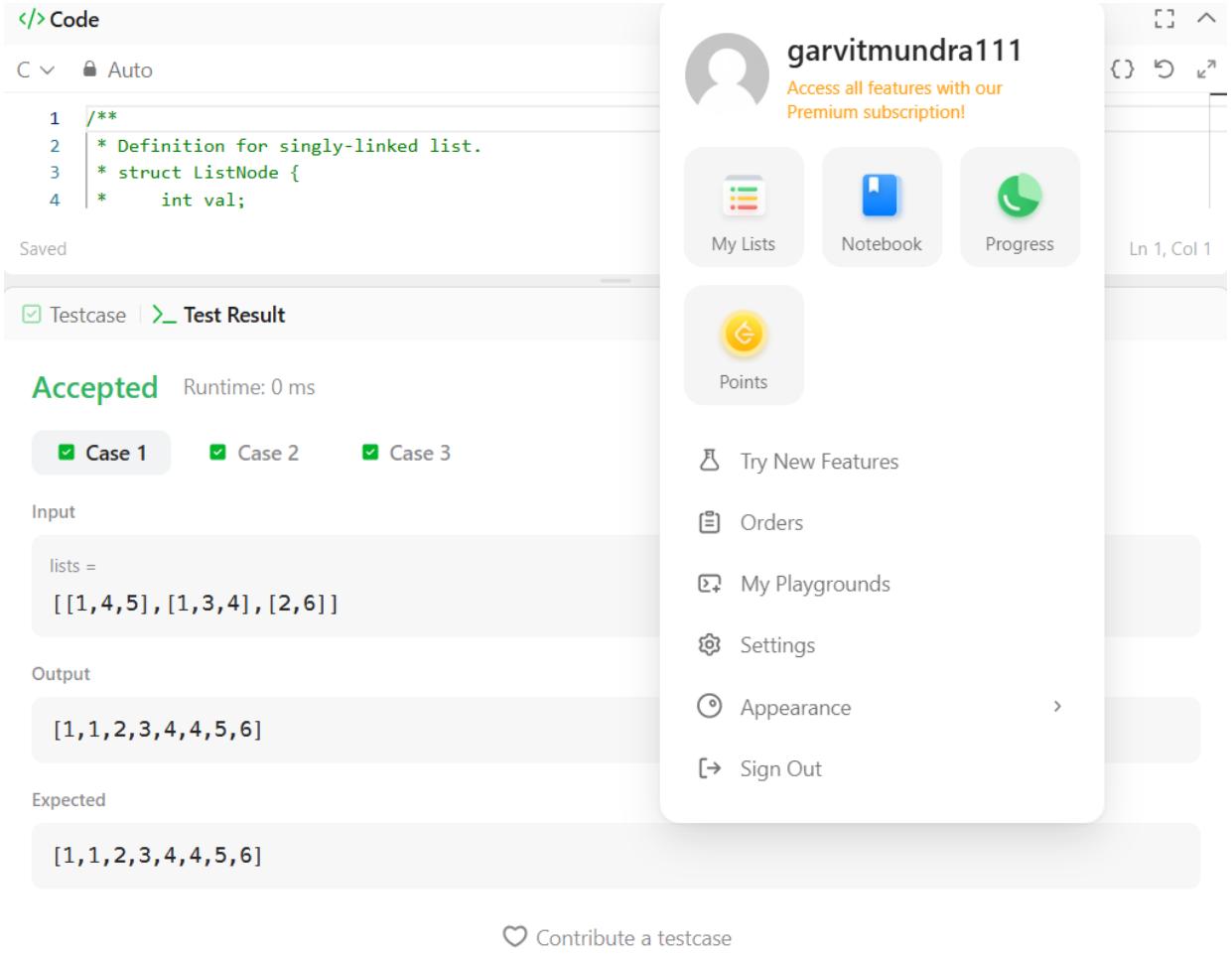
Leetcode link: https://leetcode.com/problems/merge-k-sorted-lists/description/

CODE:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode*merger(struct ListNode* lm,struct ListNode* l){
    struct ListNode*p1=lm;
    struct ListNode*p2=l;
    struct ListNode*head;
    struct ListNode*last;
    struct ListNode*dummy=(struct ListNode*)malloc(sizeof(struct
ListNode)*1);
    dummy->next=NULL;
    head=dummy;
    last=dummy;
    while(p1!=NULL&&p2!=NULL){
        if(p1->val<p2->val){
            last->next=p1;
            last=p1;
            p1=p1->next;
            last->next=NULL;

        }
        else{
            last->next=p2;
            last=p2;
            p2=p2->next;
            last->next=NULL;
        }
    }
    if(p1!=NULL){
        last->next=p1;
    }
    if(p2!=NULL){
        last->next=p2;
```

```
    }
    return head->next;
}
struct ListNode* mergeKLists(struct ListNode** lists, int listsSize)
{
    if(listsSize==0){
        return NULL;
    }
struct ListNode*res=lists[0];
// struct ListNode*last=res;
    for(int i=1;i<listsSize;i++){
        res=merger(res,lists[i]);
    }
    return res;
}
```

TESTCASES:


Code
C
Auto
1 /**
2 * Definition for singly-linked list.
3 * struct ListNode {
4 * int val;
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
lists =
[[1,4,5],[1,3,4],[2,6]]
Output
[1,1,2,3,4,4,5,6]
Expected
[1,1,2,3,4,4,5,6]
Contribute a testcase
garvitmundra111
Access all features with our Premium subscription!
My Lists
Notebook
Progress
Points
Try New Features
Orders
My Playgrounds
Settings
Appearance
Sign Out

Coding challenge 3:

You have k lists of sorted integers in **non-decreasing order**. Find the **smallest** range that includes at least one number from each of the k lists.

We define the range [a, b] is smaller than range [c, d] if b - a < d - c **or** a < c if b - a == d - c.

Example 1:

Input: nums = [[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]

Output: [20,24]

Explanation:

List 1: [4, 10, 15, 24,26], 24 is in range [20,24].

List 2: [0, 9, 12, 20], 20 is in range [20,24].

List 3: [5, 18, 22, 30], 22 is in range [20,24].

Example 2:

Input: nums = [[1,2,3],[1,2,3],[1,2,3]]

Output: [1,1]

Constraints:

- nums.length == k
- 1 <= k <= 3500
- 1 <= nums[i].length <= 50
- -10^5 <= nums[i][j] <= 10^5
- nums[i] is sorted in **non-decreasing** order.

Leet code link : https://leetcode.com/problems/smallest-range-covering-elements-from-k-lists/description/

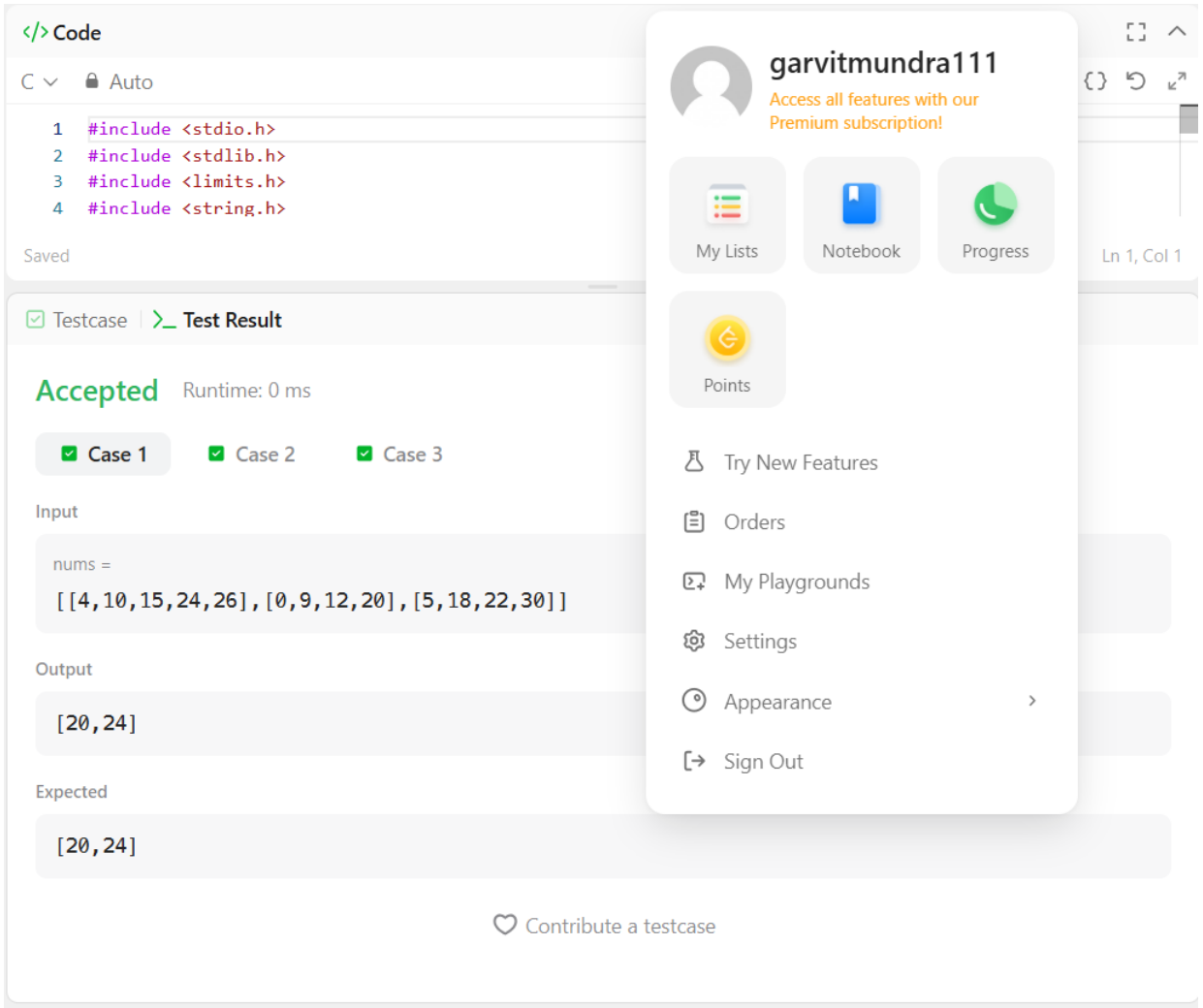
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

// Define a pair structure to hold value and index
typedef struct {
    int value;
    int index;
} Pair;

// Comparator function for qsort
int comparePairs(const void* a, const void* b) {
    return ((Pair*)a)->value - ((Pair*)b)->value;
}

// Function to find the smallest range
int* smallestRange(int** nums, int numsSize, int* numsColSize,
int* returnSize) {
    int n = 0;
    for (int i = 0; i < numsSize; i++) {
        n += numsColSize[i];
    }

    Pair* t = (Pair*)malloc(n * sizeof(Pair));
    int k = numsSize;
    int index = 0;

    // Populate the t array with value-index pairs
    for (int i = 0; i < k; ++i) {
        for (int j = 0; j < numsColSize[i]; ++j) {
            t[index].value = nums[i][j];
            t[index].index = i;
            ++index;
        }
    }
```

```
    // Sort the array based on values
    qsort(t, n, sizeof(Pair), comparePairs);

    int* ans = (int*)malloc(2 * sizeof(int));
    ans[0] = -1000000;
    ans[1] = 1000000;

    int* cnt = (int*)calloc(k, sizeof(int));
    int j = 0, distinctCount = 0;

    // Sliding window to find the smallest range
    for (int i = 0; i < n; ++i) {
        int value = t[i].value;
        int groupIndex = t[i].index;

        if (cnt[groupIndex]++ == 0) {
            ++distinctCount;
        }

        while (distinctCount == k) {
            int startValue = t[j].value;
            int startGroupIndex = t[j].index;

            int rangeLength = value - startValue;
            int currentRangeLength = ans[1] - ans[0];

            if (rangeLength < currentRangeLength ||
                (rangeLength == currentRangeLength && startValue <
ans[0])) {
                ans[0] = startValue;
                ans[1] = value;
            }

            if (--cnt[startGroupIndex] == 0) {
                --distinctCount;
            }
            ++j;
        }
```

```
    }

    free(t);
    free(cnt);

    *returnSize = 2;
    return ans;
}
```

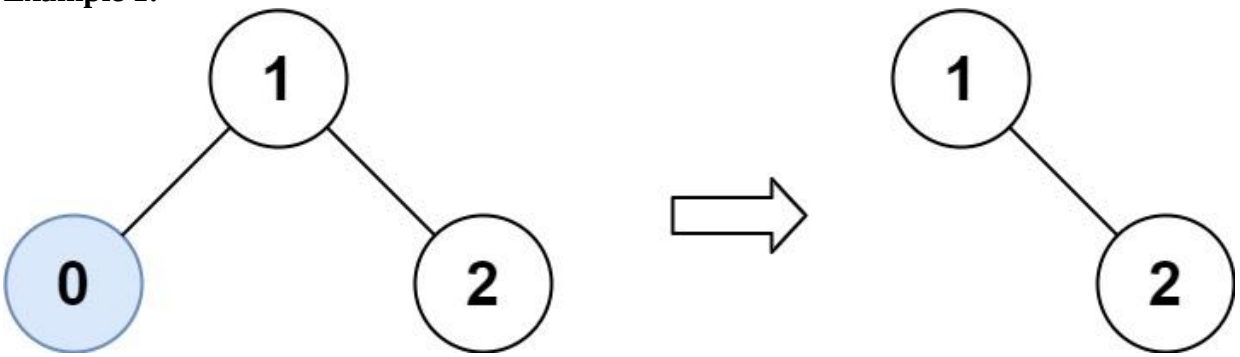
TESTCASES:


Code
C
Auto
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4 #include <string.h>
Saved
Ln 1, Col 1
garvitmundra111
Access all features with our Premium subscription!
My Lists
Notebook
Progress
Points
Try New Features
Orders
My Playgrounds
Settings
Appearance
Sign Out
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]
Output
[20,24]
Expected
[20,24]
Contribute a testcase

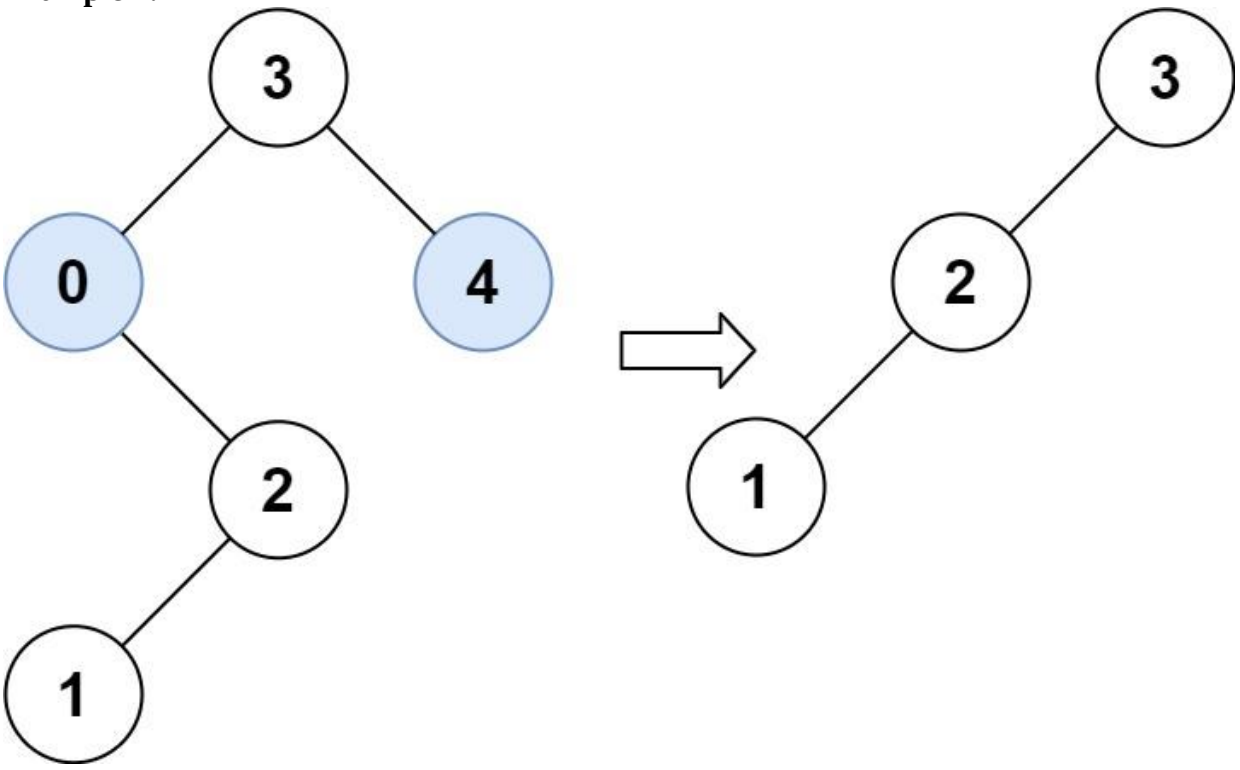
Coding challenge 4:

Given the root of a binary search tree and the lowest and highest boundaries as low and high, trim the tree so that all its elements lies in [low, high]. Trimming the tree should **not** change the relative structure of the elements that will remain in the tree (i.e., any node's descendant should remain a descendant). It can be proven that there is a **unique answer**.
Return *the root of the trimmed binary search tree*. Note that the root may change depending on the given bounds.

Example 1:



Input: root = [1,0,2], low = 1, high = 2
Output: [1,null,2]
Example 2:



Input: root = [3,0,4,null,2,null,null,1], low = 1, high = 3

Output: [3,2,null,1]

Constraints:

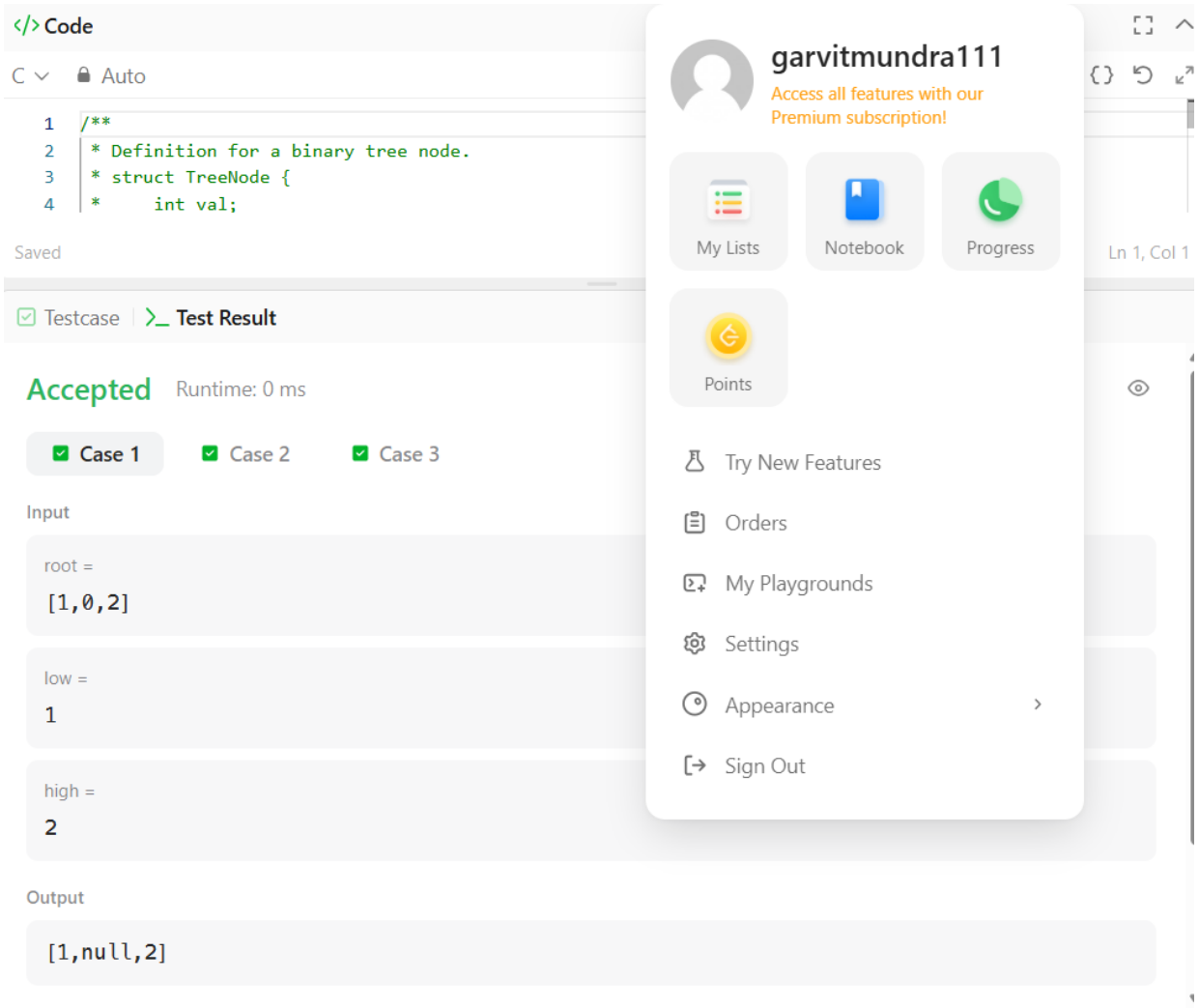
- The number of nodes in the tree is in the range [1, 10^4].
- 0 <= Node.val <= 10^4
- The value of each node in the tree is **unique**.
- root is guaranteed to be a valid binary search tree.
- 0 <= low <= high <= 10^4

Leet code link :
https://leetcode.com/problems/trim-a-binary-search-tree/

CODE:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* trimBST(struct TreeNode* root, int low, int high){
    if ( ! root ) return root;
    if ( root->val < low ) return trimBST(root->right, low, high);
    if ( root->val > high ) return trimBST(root->left, low, high);
    root->left = trimBST(root->left, low, high);
    root->right = trimBST(root->right, low, high);
    return root;
}
```

TESTCASES:


Code
C
Auto
1 /**
2 * Definition for a binary tree node.
3 * struct TreeNode {
4 * int val;
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[1,0,2]
low =
1
high =
2
Output
[1,null,2]
garvitmundra111
Access all features with our Premium subscription!
My Lists
Notebook
Progress
Points
Try New Features
Orders
My Playgrounds
Settings
Appearance
Sign Out

Coding challenge 5:

An array is squareful if the sum of every pair of adjacent elements is a perfect square.

Given an integer array nums, return *the number of permutations of* nums *that are squareful*.

Two permutations perm1 and perm2 are different if there is some index i such that perm1[i] != perm2[i].

Example 1:

Input: nums = [1,17,8]

Output: 2

Explanation: [1,8,17] and [17,8,1] are the valid permutations.

Example 2:

Input: nums = [2,2,2]

Output: 1

Constraints:

- **1 <= nums.length <= 12**
- **0 <= nums[i] <= 10^9**

Leetcode link : https://leetcode.com/problems/number-of-squareful-arrays/description/

CODE:

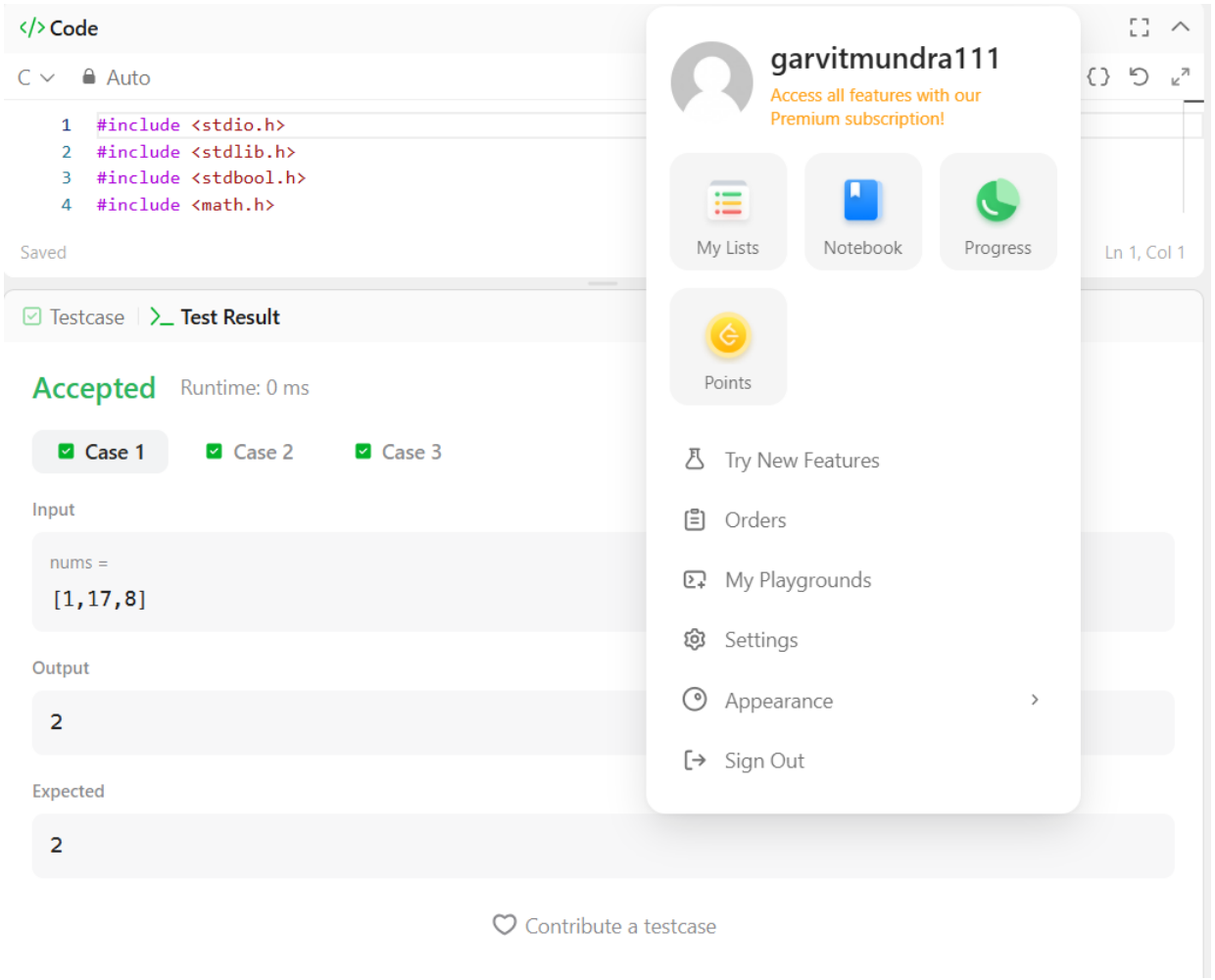
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

// Global variable to store the final count
int global_count = 0;

/**
 * @brief Checks if the sum of two numbers is a perfect square.
 * @param a First number.
 * @param b Second number.
 * @return true if a + b is a perfect square, false otherwise.
 */
bool is_perfect_square(int a, int b) {
    long long sum = (long long)a + b;
    if (sum < 0) return false;
    long long temp = round(sqrt(sum));
    return (temp * temp) == sum;
}

/**
 * Comparison function for qsort (ascending order).
 */
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

/**
 * @brief Recursive backtracking function using a 'used' array
(Standard approach).
 *
 * @param nums The original sorted array.
 * @param numsSize The size of the array.
 * @param used Boolean array to track which elements are
currently used.
 * @param currentPermSize The number of elements currently in
```

```
the permutation (0 to numsSize).
 * @param lastElement The last element added to the permutation.
 */
void backtrack(int* nums, int numsSize, bool* used, int
currentPermSize, int lastElement) {
    // Base Case: A complete permutation is formed
    if (currentPermSize == numsSize) {
        global_count++;
        return;
    }

    for (int i = 0; i < numsSize; i++) {

        // 1. Skip if element is already used
        if (used[i]) {
            continue;
        }

        // 2. Squareful check (Adjacency Rule)
        // Only check the rule if we are placing the second or later
element
        if (currentPermSize > 0) {
            if (!is_perfect_square(lastElement, nums[i])) {
                continue; // This element cannot be placed next
            }
        }

        // 3. Duplicate handling (The core fix for Permutations II style
problems)
        // If the current element is a duplicate of the previous one
(nums[i] == nums[i-1])
        // AND the previous one was NOT used (!used[i-1]), skip the
current element.
        // This prevents generating identical permutations like (1_a, 8,
...) and (1_b, 8, ...)
        if (i > 0 && nums[i] == nums[i-1] && !used[i-1]) {
            continue;
        }
```

```
        // --- Choose (Action) ---
        used[i] = true;

        // --- Explore (Recurse) ---
        backtrack(nums, numsSize, used, currentPermSize + 1,
nums[i]);

        // --- Unchoose (Backtrack) ---
        used[i] = false;
    }
}

/**
 * @brief Main function for LeetCode problem 996.
 * * @param nums The integer array.
 * @param numsSize The size of the array.
 * @return The number of squareful permutations.
 */
int numSquarefulPerms(int* nums, int numsSize) {
    if (numsSize == 0) return 0;

    // 1. Sort the array: ESSENTIAL for duplicate handling
    qsort(nums, numsSize, sizeof(int), compare);

    // 2. Initialize tracking variables
    bool* used = (bool*)calloc(numsSize, sizeof(bool));
    global_count = 0;

    // 3. Start the backtracking process
    // currentPermSize = 0 (starting empty), lastElement is irrelevant
at start
    backtrack(nums, numsSize, used, 0, 0);

    // 4. Clean up and return
    free(used);
    return global_count;
}

// NOTE: Submit only the functions above.
```

// Remove any 'main' function or test code.

TESTCASES:


Code
C
Auto
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <math.h>
Saved
Ln 1, Col 1
garvitmundra111
Access all features with our Premium subscription!
My Lists
Notebook
Progress
Points
Try New Features
Orders
My Playgrounds
Settings
Appearance
Sign Out
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[1,17,8]
Output
2
Expected
2
Contribute a testcase