

Harsh Pandey

22BCS17030

IOT-609/B

Advance Programming

Assignment 4

## 1. Longest Nice Substring:

**CODE:**

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        if (s.size() < 2) return "";
        unordered_set<char> charSet(s.begin(), s.end());
        for (int i = 0; i < s.size(); i++) {
            if (charSet.count(tolower(s[i])) == 0 || charSet.count(toupper(s
            [i])) == 0)
            {
                string left = longestNiceSubstring(s.substr(0, i));
                string right = longestNiceSubstring(s.substr(i + 1));
                return left.size() >= right.size() ? left : right;
            }
        }
        return s;
    }
};
```

The screenshot shows a code editor interface with a problem list on the left and a code editor on the right. The problem list shows three submissions: two 'Accepted' and one 'Compile Error'. The code editor shows the C++ code for the 'Longest Nice Substring' problem. The code is as follows:

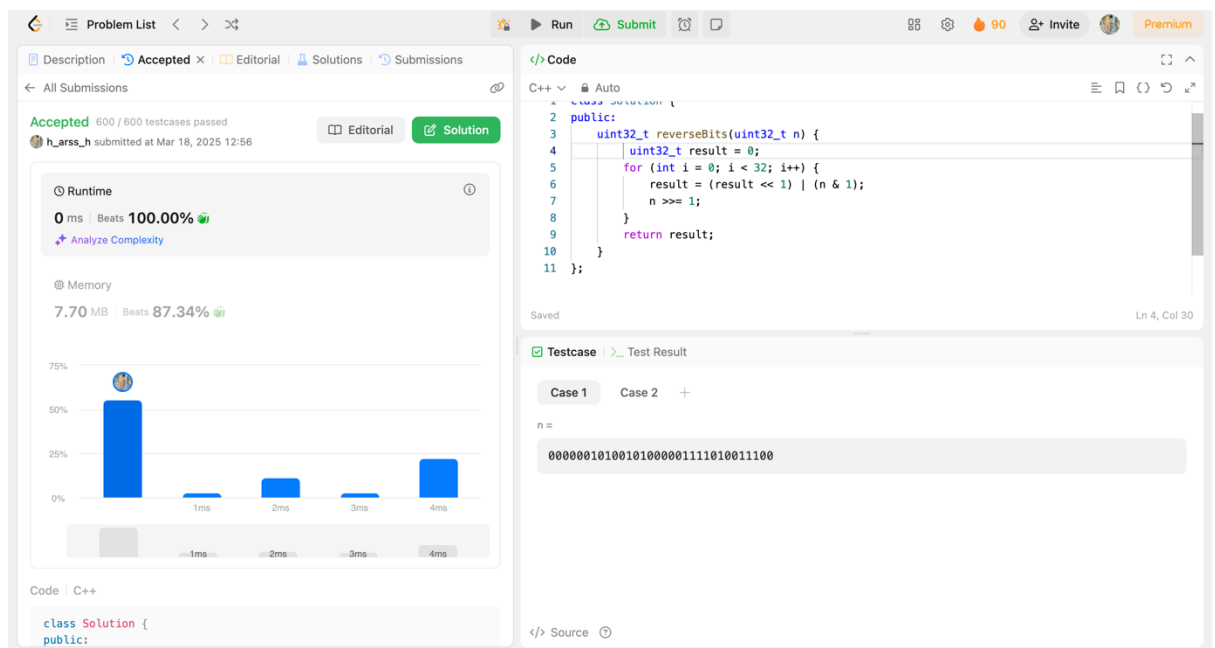
```
1 class Solution {
2 public:
3     string longestNiceSubstring(string s) {
4         if (s.size() < 2) return "";
5         unordered_set<char> charSet(s.begin(), s.end());
6         for (int i = 0; i < s.size(); i++) {
7             if (charSet.count(tolower(s[i])) == 0 || charSet.count(toupper(s
8             [i])) == 0)
9             {
10                string left = longestNiceSubstring(s.substr(0, i));
11                string right = longestNiceSubstring(s.substr(i + 1));
```

Below the code editor, the 'Test Result' section shows the test case 'Case 1' as 'Accepted' with a runtime of 0 ms. The input is 's = "YazaAay"' and the output is 'aAa'.

## 2. Reverse Bits:

### CODE:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```



## 3. Number of 1 Bits:

### CODE:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
```

```

        count += (n & 1); // Check last bit
        n >>= 1; // Shift right
    }
    return count;
}
};

```

The screenshot displays a LeetCode submission interface. On the left, the 'Runtime' section shows a bar chart where the user's solution is the fastest (0ms) and beats 100.00% of other submissions. The 'Memory' section shows 8.20 MB usage, beating 80.26% of others. The 'Code' section on the right shows the following C++ code:

```

1 class Solution {
2 public:
3     int hammingWeight(int n) {
4         int count = 0;
5         while (n != 0) {
6             count += (n & 1); // Check last bit
7             n >>= 1; // Shift right
8         }
9         return count;
10    }
11 };

```

Below the code, the 'Testcase' section shows a single test case with input `n = 11`.

#### 4. Maximum Subarray:

**CODE:**

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};

```

The screenshot displays a LeetCode submission for the 'Maximum Subarray' problem. The submission is 'Accepted' with a runtime of 0 ms and memory of 71.7 MB. The code implements Kadane's algorithm. The test case shows an input array [-2, 1, -3, 4, -1, 2, 1, -5, 4] and an output of 6.

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        //this is kadane's algo and it is optimized approach. as time=O(n).
        int maxSum = nums[0];
        int currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```

Testcase: Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums = [-2,1,-3,4,-1,2,1,-5,4]

Output

6

Expected

## 5. Search a 2D Matrix II:

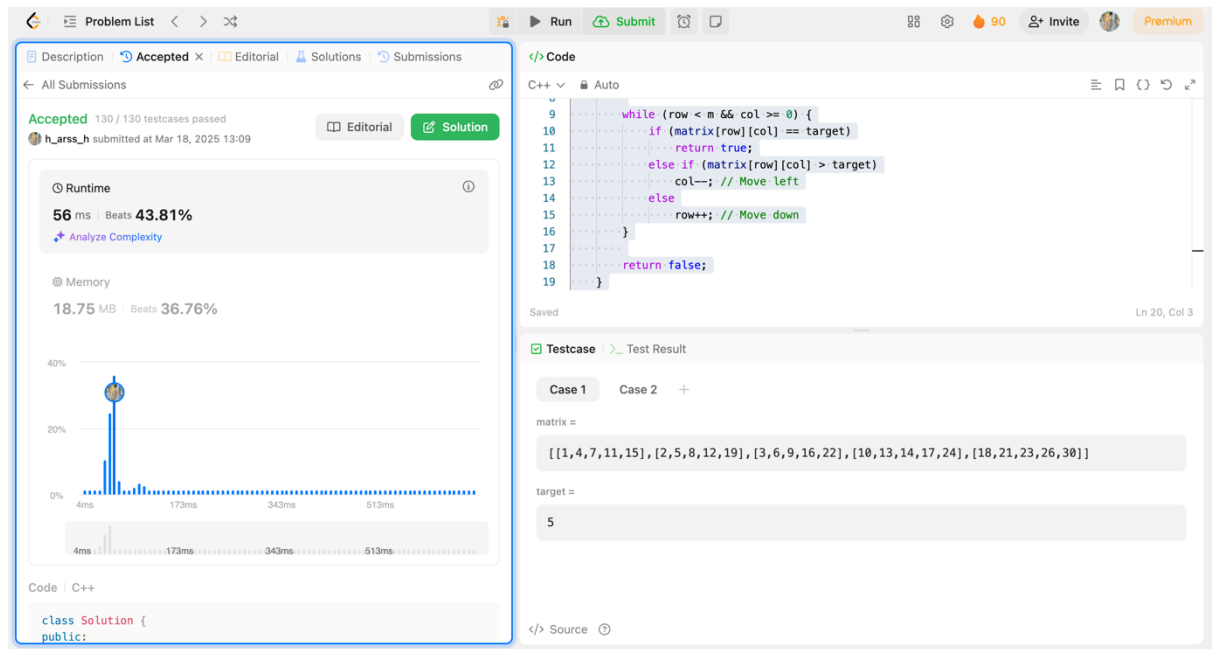
### CODE:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size();
        int n = matrix[0].size();

        int row = 0, col = n - 1; // Start from top-right

        while (row < m && col >= 0) {
            if (matrix[row][col] == target)
                return true;
            else if (matrix[row][col] > target)
                col--; // Move left
            else
                row++; // Move down
        }

        return false;
    }
};
```



## 6. Super Pow:

### CODE:

```
class Solution {
```

```
public:
```

```
    const int MOD = 1337;
```

```
    int modPow(int x, int y, int mod) {
```

```
        int result = 1;
```

```
        x = x % mod;
```

```
        while (y > 0) {
```

```
            if (y % 2 == 1) {
```

```
                result = (result * x) % mod;
```

```
            }
```

```
            x = (x * x) % mod; // Square x
```

```
            y /= 2; // Reduce y
```

```
        }
```

```
        return result;
```

```
    }
```

```
    int superPow(int a, vector<int>& b) {
```

```
        if (b.empty()) return 1; // Base case
```

```
        int lastDigit = b.back();
```

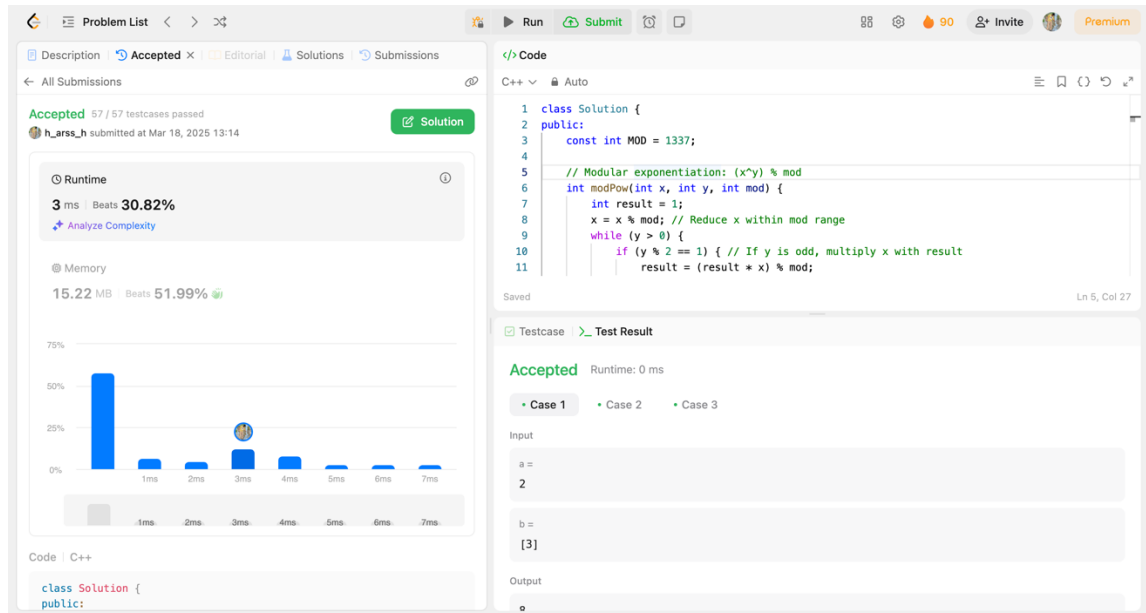
```
        b.pop_back(); // Remove last digit
```

```
        int part1 = modPow(superPow(a, b), 10, MOD);
```

```

    int part2 = modPow(a, lastDigit, MOD);
    return (part1 * part2) % MOD;
}
};

```



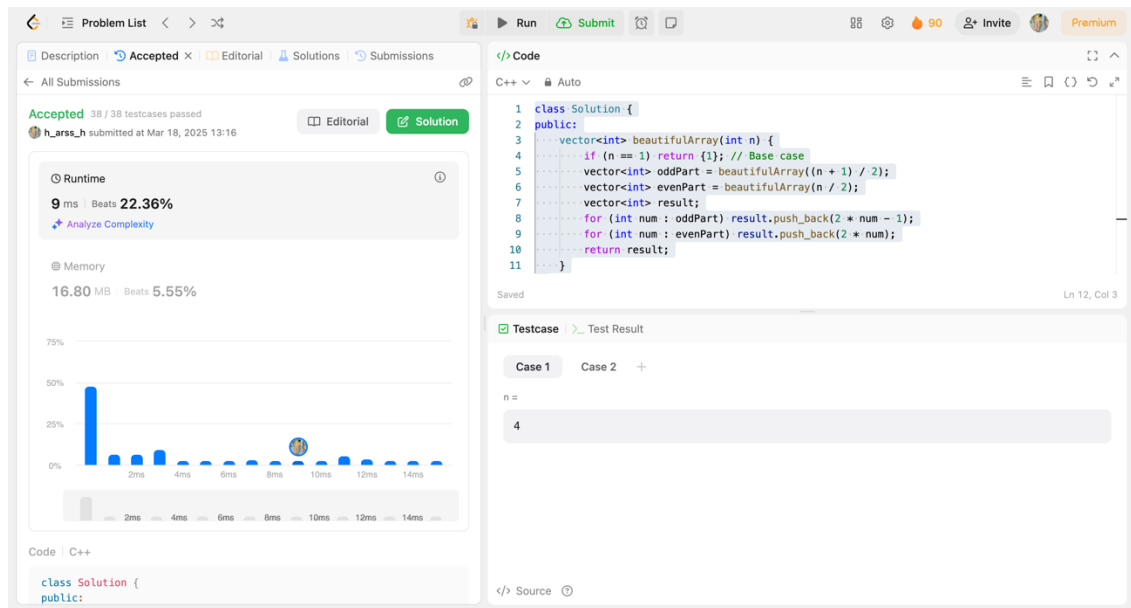
## 7. Beautiful Array:

### CODE:

```

class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1}; // Base case
        vector<int> oddPart = beautifulArray((n + 1) / 2);
        vector<int> evenPart = beautifulArray(n / 2);
        vector<int> result;
        for (int num : oddPart) result.push_back(2 * num - 1);
        for (int num : evenPart) result.push_back(2 * num);
        return result;
    }
};

```



## 8. The Skyline Problem:

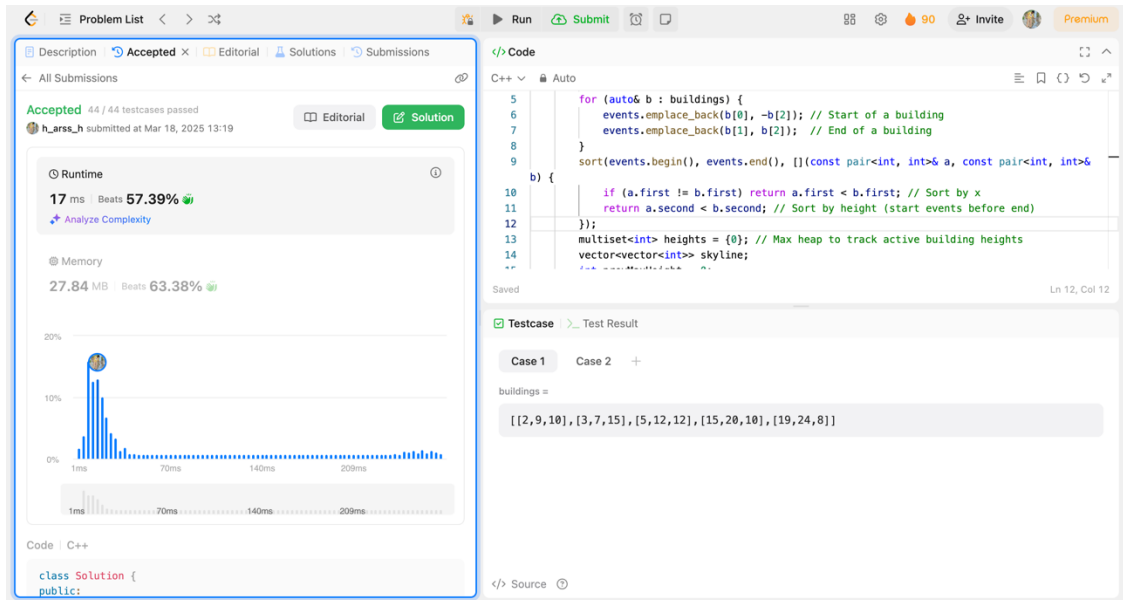
### CODE:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events; // Store (x, height)
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Start of a building
            events.emplace_back(b[1], b[2]); // End of a building
        }
        sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
            if (a.first != b.first) return a.first < b.first; // Sort by x
            return a.second < b.second; // Sort by height (start events before end)
        });
        multiset<int> heights = {0}; // Max heap to track active building heights
        vector<vector<int>> skyline;
        int prevMaxHeight = 0;
        for (auto& e : events) {
            int x = e.first, h = e.second;
            if (h < 0) heights.insert(-h); // Building starts: add height
            else heights.erase(heights.find(h)); // Building ends: remove height
            int currMaxHeight = *heights.rbegin(); // Get max height
            if (currMaxHeight != prevMaxHeight) { // If height changes, record key point
                skyline.push_back({x, currMaxHeight});
                prevMaxHeight = currMaxHeight;
            }
        }
        return skyline;
    }
};
```

```

    }
}
return skyline;
}
};

```



## 9. Reverse Pairs:

### CODE:

```
class Solution {
```

```
public:
```

```

    int reversePairs(vector<int>& nums) {
        return mergeSort(nums, 0, nums.size() - 1);
    }

```

```
private:
```

```

    int mergeSort(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;

        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

        // Count valid reverse pairs
        count += countPairs(nums, left, mid, right);

        // Merge the sorted halves
        merge(nums, left, mid, right);
    }

```



```

        return count;
    }

    int countPairs(vector<int>& nums, int left, int mid, int right) {
        int count = 0;
        int j = mid + 1;

        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2LL * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }

        return count;
    }

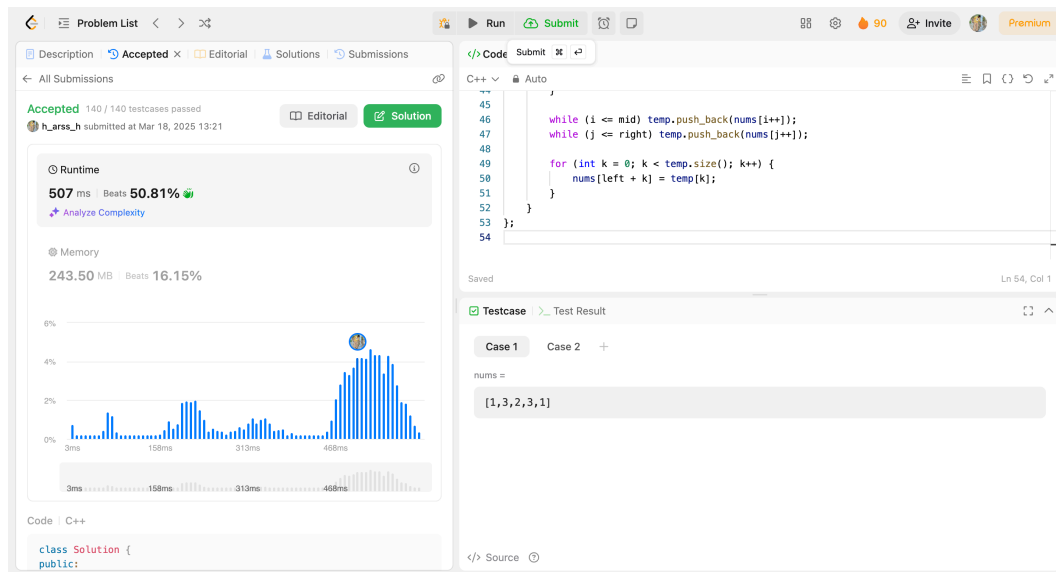
    void merge(vector<int>& nums, int left, int mid, int right) {
        vector<int> temp;
        int i = left, j = mid + 1;

        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) temp.push_back(nums[i++]);
            else temp.push_back(nums[j++]);
        }

        while (i <= mid) temp.push_back(nums[i++]);
        while (j <= right) temp.push_back(nums[j++]);

        for (int k = 0; k < temp.size(); k++) {
            nums[left + k] = temp[k];
        }
    }
};

```



## 10. Longest Increasing Subsequence II:

### CODE:

```
class SegmentTree{
public:
    int leftIndex;
    int rightIndex;
    SegmentTree* left;
    SegmentTree* right;
    int maxNum;
    SegmentTree(int leftl,int rightl,int val){
        leftIndex=leftl;
        rightIndex=rightl;
        maxNum=val;
        left=NULL;
        right=NULL;
    }

    void updateTree(int index,int val,SegmentTree* root){
        if(root->leftIndex==root->rightIndex){
            root->maxNum=val;
            return;
        }

        int midIndex=(root->leftIndex+root->rightIndex)/2;
        if(midIndex>=index){
            updateTree(index,val,root->left);
        } else {
```

```

        updateTree(index,val,root->right);
    }
    root->maxNum=max(root->left->maxNum,root->right->maxNum);
    return;
}

int query(int leftl,int rightl,SegmentTree* root){
    if(root->rightIndex==rightl && root->leftIndex==leftl)
        return root->maxNum;

    if(leftl>rightl){
        return -1;
    }

    int midIndex=(root->leftIndex+root->rightIndex)/2;
    int ans=0;
    if(leftl<=midIndex && midIndex<=rightl){
        ans=max(ans,max(query(leftl,midIndex,root->
>left),query(midIndex+1,rightl,root->right)));
    } else if(midIndex<leftl) {
        ans=max(ans,query(leftl,rightl,root->right));
    } else {
        ans=max(ans,query(leftl,rightl,root->left));
    }
    return ans;
}
};

SegmentTree* construct(int leftl,int rightl){
    if(leftl==rightl){
        return new SegmentTree(leftl,rightl,-1);
    }

    int midIndex=(leftl+rightl)/2;
    SegmentTree* root=new SegmentTree(leftl,rightl,0);
    SegmentTree* leftTree=construct(leftl,midIndex);
    SegmentTree* rightTree=construct(midIndex+1,rightl);
    root->left=leftTree;
    root->right=rightTree;
    root->maxNum=max(leftTree->maxNum,rightTree->maxNum);

```

```

        return root;
    }

    class Solution {
    public:
        int lengthOfLIS(vector<int>& nums, int k) {
            int maxN=-1;
            for(auto n:nums){
                maxN=max(maxN,n);
            }
            stack<int> st;
            SegmentTree* root;
            root=construct(0,maxN+k);
            int ans=1;
            for(int i=nums.size()-1;i>=0;i--){
                int n=nums[i];
                while(!st.empty() && (st.top()<=n || st.top()>n+k)){
                    st.pop();
                }
                st.push(n);
                int l=root->query(n+1,n+k,root)+1;
                ans=max(ans,l);
                root->updateTree(n,l,root);
            }
            return ans;
        }
    };

```

Problem List < > > >

Description Accepted x Editorial Solutions Submissions

All Submissions

Accepted 84 / 84 testcases passed

h\_arss\_h submitted at Mar 18, 2025 13:24

Solution

Runtime 730 ms Beats 5.14%

Analyze Complexity

Memory 316.81 MB Beats 5.35%

10% 5% 0%

11ms 72ms 133ms 195ms 256ms 317ms 379ms

Code | C++

```

class SegmentTree{
public:

```

Testcase > Test Result

Case 1 Case 2 Case 3 +

nums =

[4, 2, 1, 4, 3, 4, 5, 8, 15]

k =

3

</ Source