



## ASSIGNMENT-4

**Student Name:** Rahul

**UID:** 22BCS17111

**Branch:** CSE

**Section/Group:** 22BCS\_IOT-609/B

**Semester:** 6<sup>th</sup>

**Subject Code:** 22CSP-351

**Subject Name:** Advanced Programming Lab-II

### 1. Problem Statement :

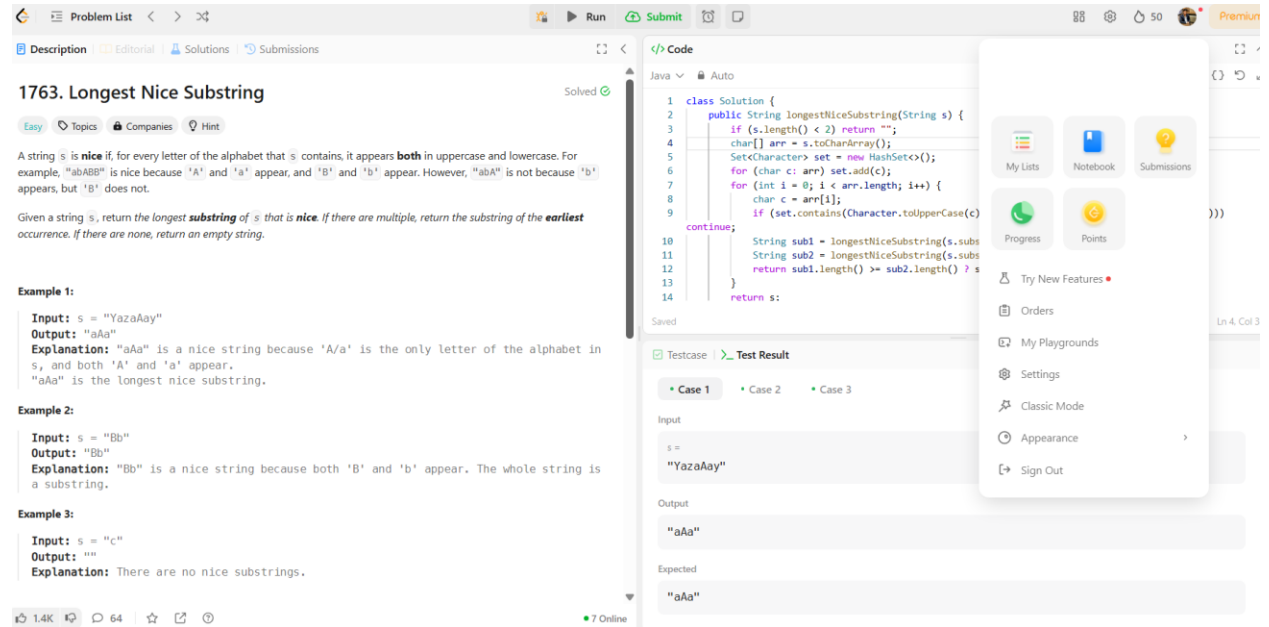
#### Longest Nice Substring

<https://leetcode.com/problems/longest-nice-substring/description/>

#### Code:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";
        char[] arr = s.toCharArray();
        Set<Character> set = new HashSet<>();
        for (char c: arr) set.add(c);
        for (int i = 0; i < arr.length; i++) {
            char c = arr[i];
            if (set.contains(Character.toUpperCase(c)) &&
                set.contains(Character.toLowerCase(c))) continue;
            String sub1 = longestNiceSubstring(s.substring(0, i));
            String sub2 = longestNiceSubstring(s.substring(i+1));
            return sub1.length() >= sub2.length() ? sub1 : sub2;
        }
        return s;
    }
}
```

## OUTPUT:



The screenshot shows the LeetCode interface for problem 1763, "Longest Nice Substring". The problem description states that a string is "nice" if every letter it contains appears both in uppercase and lowercase. The goal is to find the longest such substring, or the earliest occurrence if there are multiple. Three examples are provided: "YazaAay" returns "aAa", "Bb" returns "Bb", and "c" returns an empty string.

The solution code is written in Java and uses a recursive approach. It defines a method `longestNiceSubstring` that takes a string `s` and returns the longest nice substring. The code uses a `HashSet` to track the characters in the current substring and recursively calls itself on substrings that maintain the "nice" property.

```

1 class Solution {
2     public String longestNiceSubstring(String s) {
3         if (s.length() < 2) return "";
4         char[] arr = s.toCharArray();
5         Set<Character> set = new HashSet<>();
6         for (char c: arr) set.add(c);
7         for (int i = 0; i < arr.length; i++) {
8             char c = arr[i];
9             if (set.contains(Character.toUpperCase(c))
10                continue;
11             String sub1 = longestNiceSubstring(s.substring(i+1, s.length()));
12             String sub2 = longestNiceSubstring(s.substring(0, i));
13             return sub1.length() >= sub2.length() ? s.substring(0, i+1) : sub1;
14         }
15     }
16 }

```

The test result section shows the input "YazaAay" and the output "aAa", which matches the expected result.

## 2. Problem Statement:

### Reverse Bits

<https://leetcode.com/problems/reverse-bits/description/>

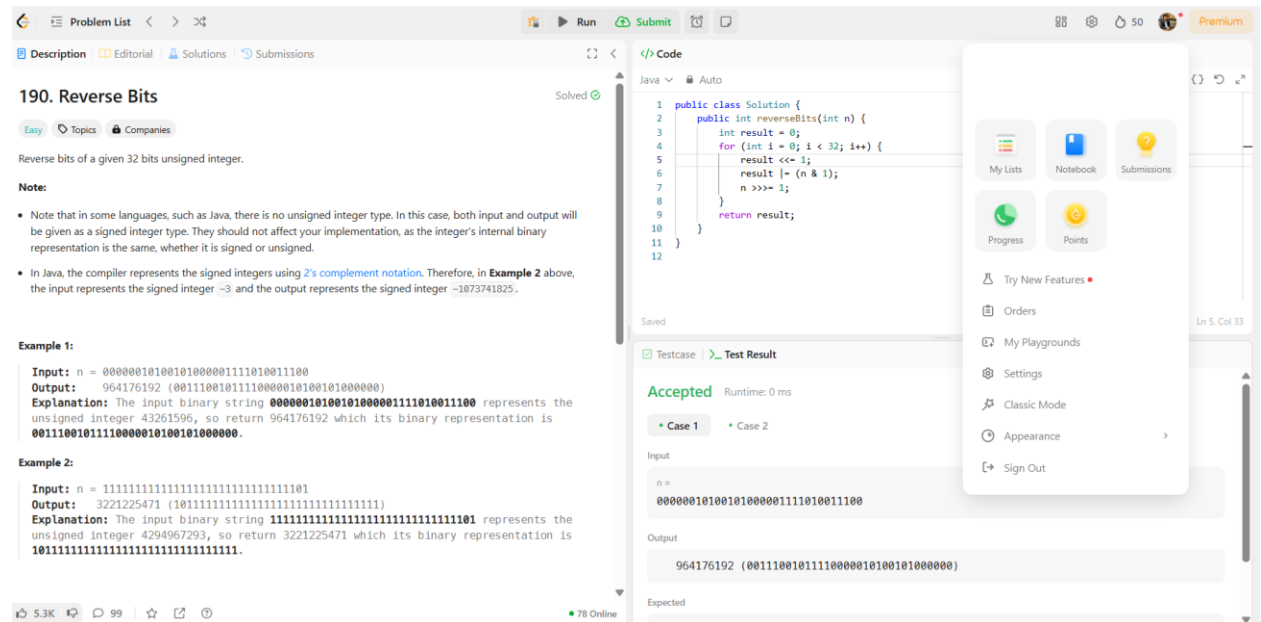
### Code:

```

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

```

## OUTPUT:



The screenshot shows a LeetCode problem titled "190. Reverse Bits" with a difficulty of "Easy". The problem description states: "Reverse bits of a given 32 bits unsigned integer." It includes a note about Java's signed integer type and 2's complement notation. Two examples are provided: Example 1 with input n = 00000010100101000001111010011100 and output 964176192, and Example 2 with input n = 11111111111111111111111111111101 and output 3221225471. The code editor on the right shows a Java solution for the reverseBits method. The test results section shows "Accepted" with a runtime of 0 ms for Case 1.

### 3. Problem Statement:

#### Number of 1 Bits

<https://leetcode.com/problems/number-of-1-bits/description/>

#### CODE:

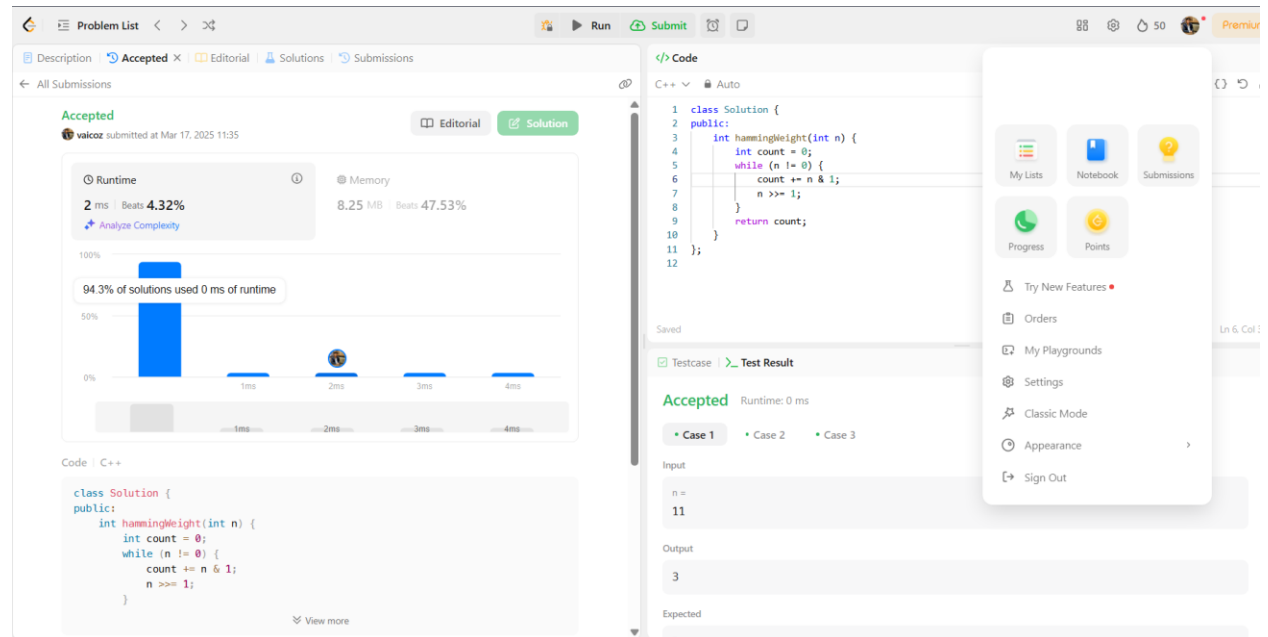
```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }
};
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## OUTPUT:



## 4. Problem Statement:

### Maximum Subarray

<https://leetcode.com/problems/maximum-subarray/description/>

## CODE:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum=0;
        int maxi=nums[0];

        for (int i=0;i<nums.size();i++){
            sum=sum+nums[i];
            maxi=max(maxi,sum);

            if (sum<0)
            {
```

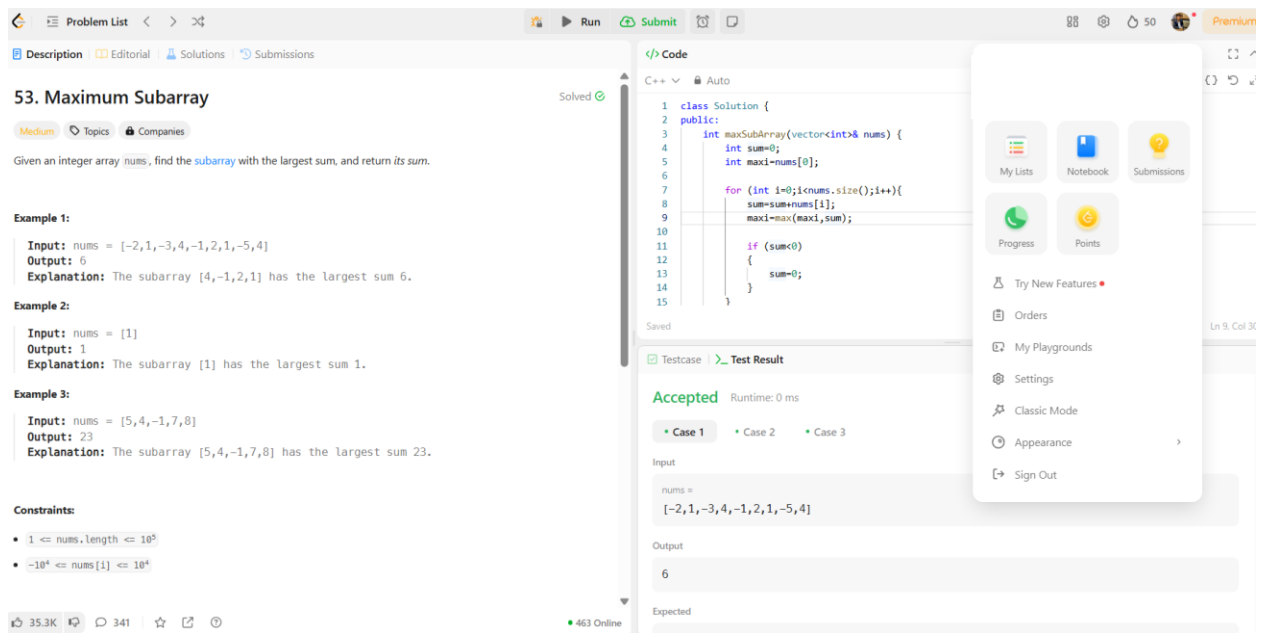
```

        sum=0;
    }
}
return maxi;

}
};

```

## OUTPUT:



The screenshot shows a LeetCode problem titled "53. Maximum Subarray" with a medium difficulty level. The problem description asks to find the subarray with the largest sum. Three examples are provided: Example 1 with input [-2,1,-3,4,-1,2,1,-5,4] and output 6; Example 2 with input [1] and output 1; Example 3 with input [5,4,-1,7,8] and output 23. Constraints specify the array length and element range. The solution code in C++ is shown on the right, implementing a Kadane's algorithm. The test results show the solution is accepted for all cases.

## 5. Problem Statement:

### Search a 2D Matrix II

<https://leetcode.com/problems/search-a-2d-matrix-ii/description/>

## CODE:

```

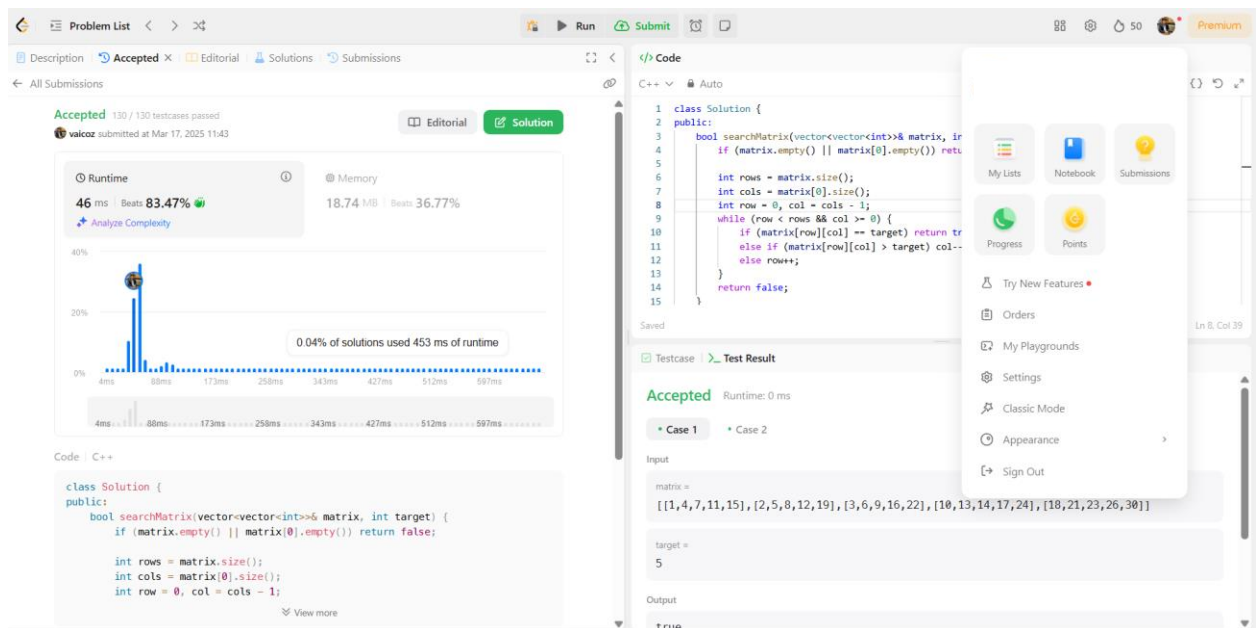
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size();
    }
};

```

```
int cols = matrix[0].size();
int row = 0, col = cols - 1;
while (row < rows && col >= 0) {
    if (matrix[row][col] == target) return true;
    else if (matrix[row][col] > target) col--;
    else row++;
}
return false;
};
```

## OUTPUT:



## 6. Problem Statement:

### Super Pow

<https://leetcode.com/problems/super-pow/description/>

### CODE:

```
class Solution {
public:
```

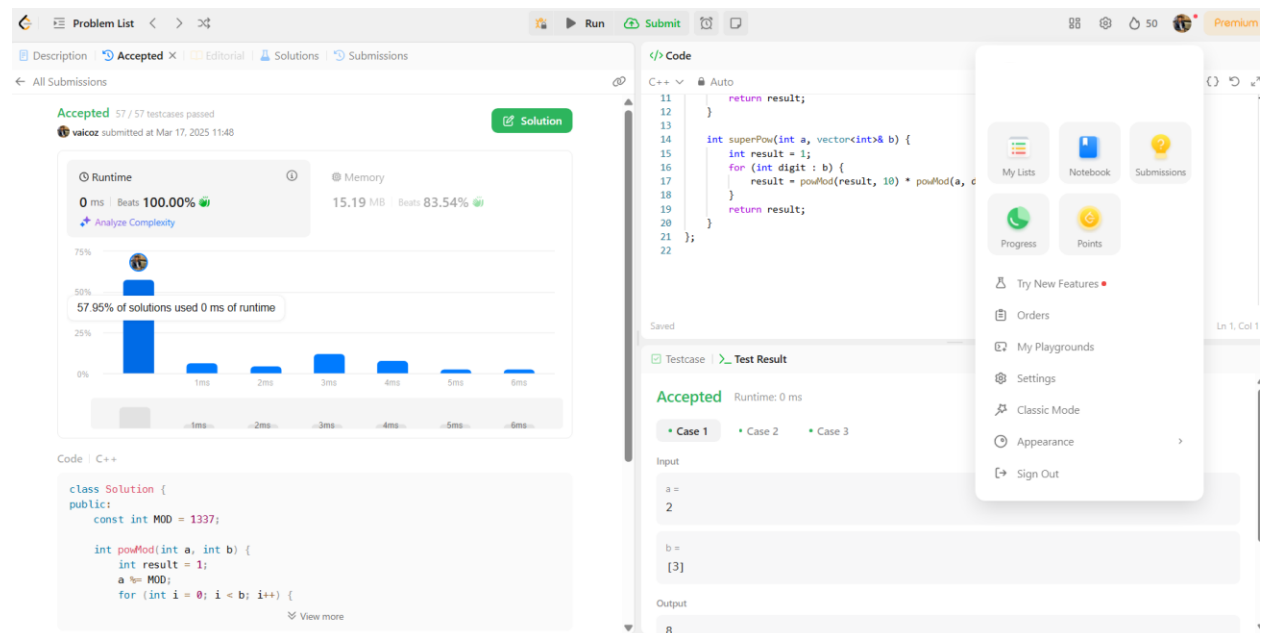
```
const int MOD = 1337;
```

```
int powMod(int a, int b) {
    int result = 1;
    a %= MOD;
    for (int i = 0; i < b; i++) {
        result = (result * a) % MOD;
    }
    return result;
}

int superPow(int a, vector<int>& b) {
    int result = 1;
    for (int digit : b) {
        result = powMod(result, 10) * powMod(a, digit) % MOD;
    }
    return result;
}

};
```

## OUTPUT:



## 7. Problem Statement:

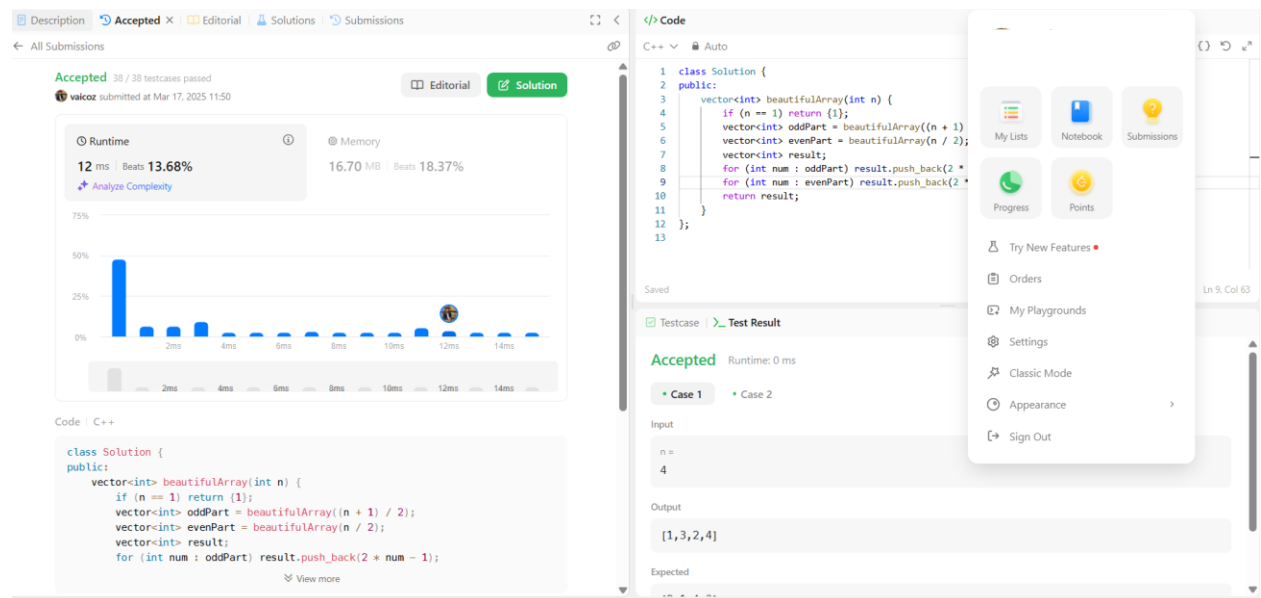
### Beautiful Array

<https://leetcode.com/problems/beautiful-array/description/>

### CODE:

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};
        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;
    }
};
```

### OUTPUT:





## 8. Problem Statement:

### The Skyline Problem

<https://leetcode.com/problems/the-skyline-problem/description/>

#### CODE:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        vector<vector<int>> result;

        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);
            events.emplace_back(b[1], b[2]);
        }

        sort(events.begin(), events.end(), [](pair<int, int>& a, pair<int, int>&
b) {
            if (a.first != b.first) return a.first < b.first;
            return a.second < b.second;
        });

        multiset<int> heights = {0};
        int prevHeight = 0;

        for (auto& e : events) {
            int x = e.first, h = e.second;

            if (h < 0) {
                heights.insert(-h);
            } else {
                heights.erase(heights.find(h));
            }
        }

        result.clear();
        for (int h : heights) {
            result.push_back({x, h});
        }

        return result;
    }
};
```

```

    }

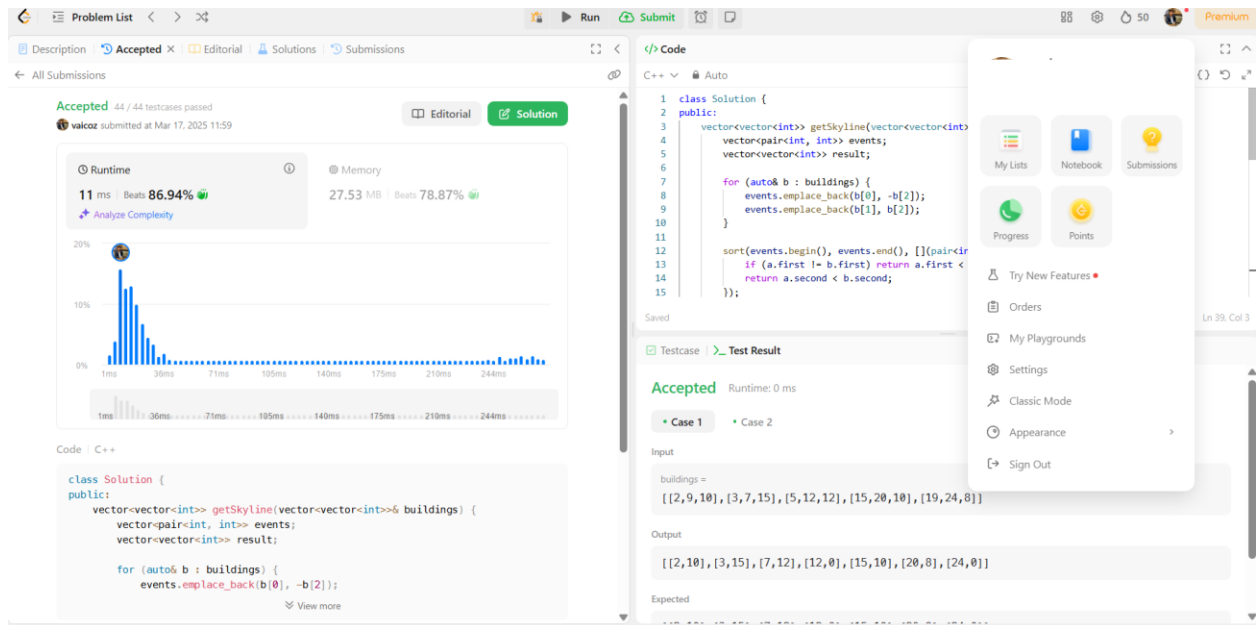
    int currHeight = *heights.rbegin();

    if (currHeight != prevHeight) {
        result.push_back({x, currHeight});
        prevHeight = currHeight;
    }
}

return result;
}
};

```

## OUTPUT:



## 9. Problem Statement:

### Reverse Pairs

<https://leetcode.com/problems/reverse-pairs/description/>

## 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);

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

        merge(nums, left, mid, right);
        return count;
    }

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

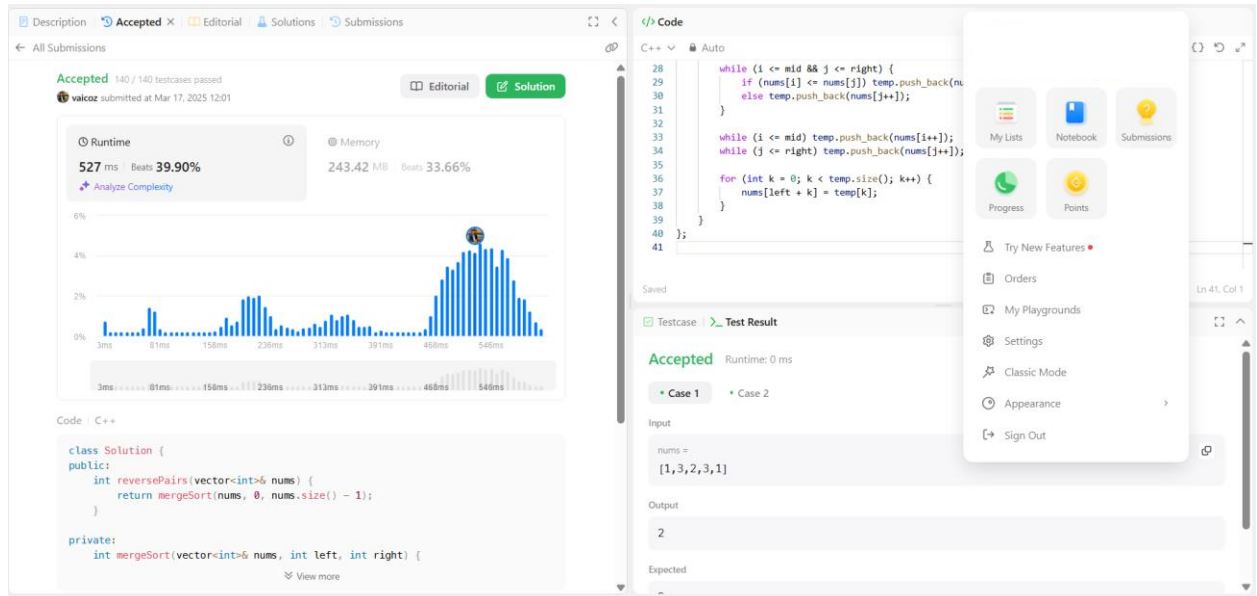
        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) temp.push_back(nums[i++]);
            else temp.push_back(nums[j++]);
        }
    }
};
```

```
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];
}
```

```
};
```

**OUTPUT:**



## 10. Problem Statement

### Longest Increasing Subsequence II:

<https://leetcode.com/problems/longest-increasing-subsequence-ii/description/>

**CODE:**

```
class SegmentTree {
    vector<int> tree;
```

```
int size;
```

```
public:
```

```
SegmentTree(int n) : size(n) {  
    tree.resize(4 * n, 0);  
}
```

```
void update(int index, int value, int node = 1, int start = 0, int end = -1)  
{  
    if (end == -1) end = size - 1;  
    if (start == end) {  
        tree[node] = value;  
        return;  
    }  
    int mid = (start + end) / 2;  
    if (index <= mid) update(index, value, 2 * node, start, mid);  
    else update(index, value, 2 * node + 1, mid + 1, end);  
    tree[node] = max(tree[2 * node], tree[2 * node + 1]);  
}
```

```
int query(int left, int right, int node = 1, int start = 0, int end = -1) {  
    if (end == -1) end = size - 1;  
    if (left > end || right < start) return 0;  
    if (left <= start && end <= right) return tree[node];  
    int mid = (start + end) / 2;  
    return max(query(left, right, 2 * node, start, mid), query(left, right, 2  
* node + 1, mid + 1, end));  
}  
};
```

```
class Solution {  
public:
```

```
int lengthOfLIS(vector<int>& nums, int k) {
    int maxVal = *max_element(nums.begin(), nums.end());
    SegmentTree segTree(maxVal + 1);
    int maxLength = 0;

    for (int num : nums) {
        int bestPrev = segTree.query(max(0, num - k), num - 1);
        int currLength = bestPrev + 1;
        segTree.update(num, currLength);
        maxLength = max(maxLength, currLength);
    }

    return maxLength;
}
```

## OUTPUT:

