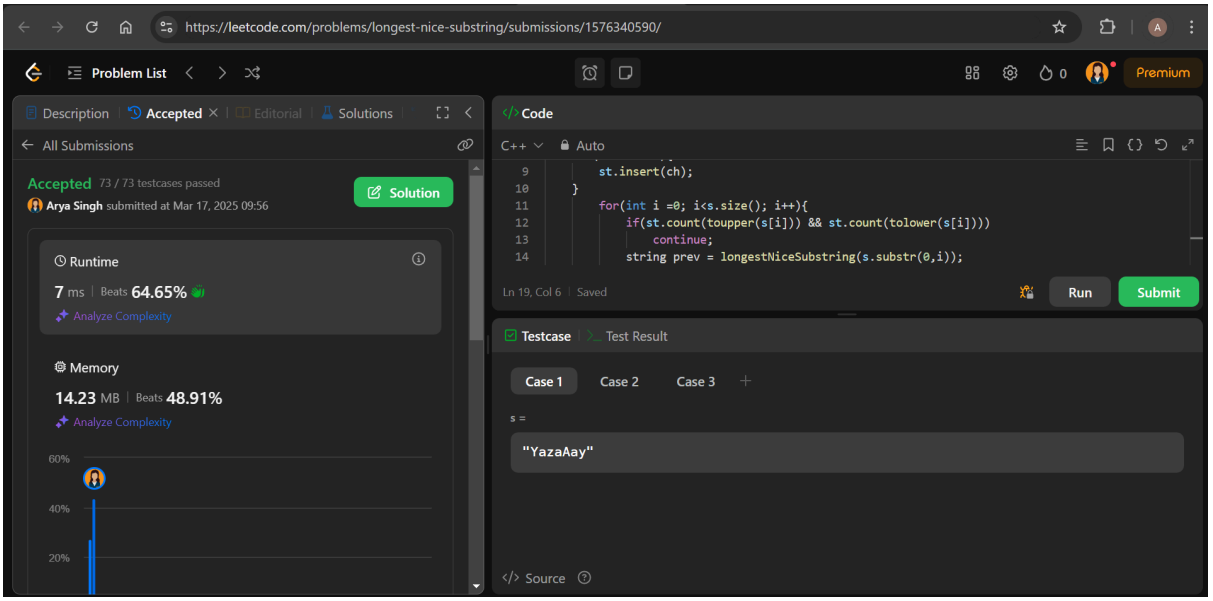
NAME : Arya Singh **UID :** 22BCS10203 **SECTION :** 607-B

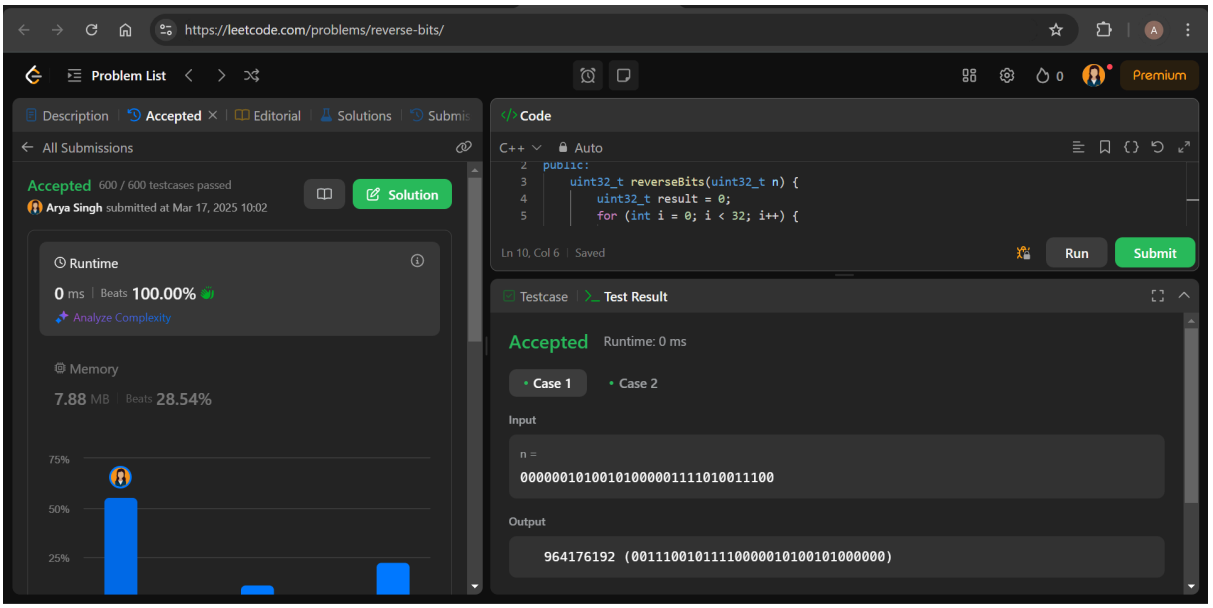
1. Longest Nice Substring

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        if(s.size() < 2)
        return "";

    unordered_set <char> st;
    for(auto ch:s){
        st.insert(ch);
    }
        for(int i =0; i<s.size(); i++){
            if(st.count(toupper(s[i])) && st.count(tolower(s[i])))
                continue;
            string prev = longestNiceSubstring(s.substr(0,i));
            string next = longestNiceSubstring(s.substr(i+1));
            return prev.size() >= next.size() ? prev : next;
    }
        return s;
    }
};
```



2. Reverse Bits

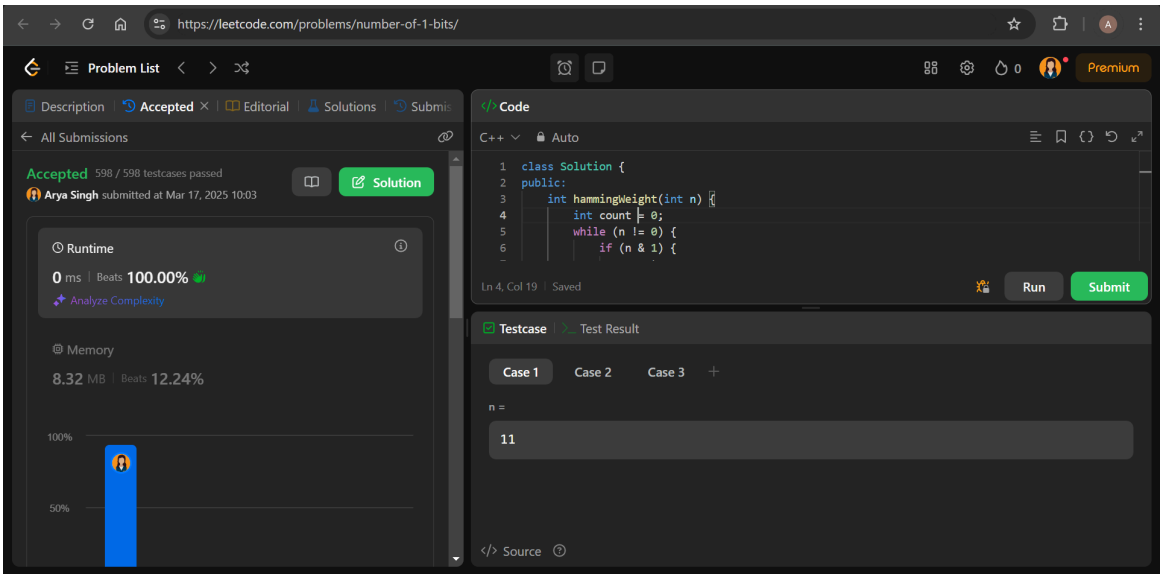
```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1); // Shift left and add last bit
            n >>= 1; // Shift right to get next bit
        }
        return result;
    }
};
```



3. Number of 1 Bits

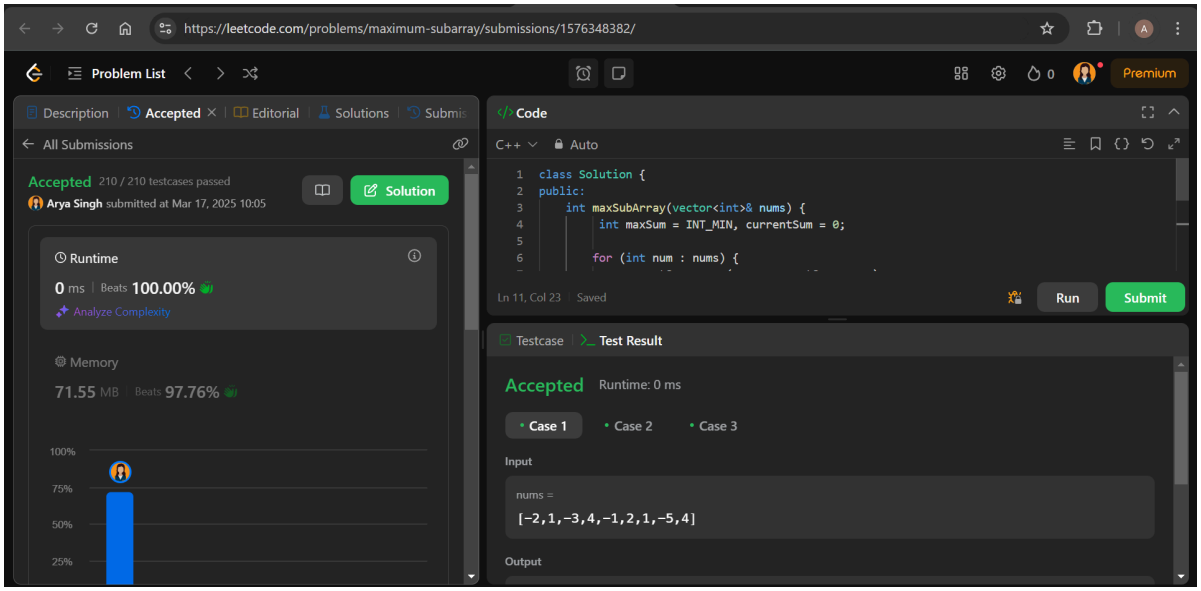
```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            if (n & 1) {
                count++;
            }
            n = n >> 1;
        }
        return count;
```

```
    }
};
```



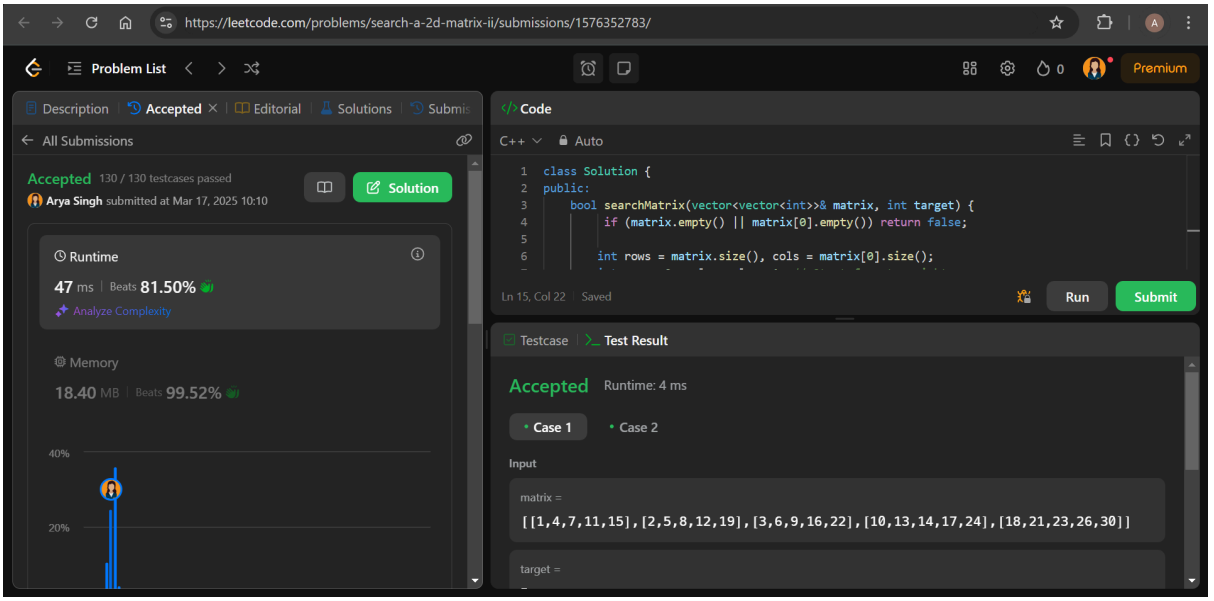
4. Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = INT_MIN, currentSum = 0;

        for (int num : nums) {
            currentSum = max(num, currentSum + num);
            maxSum = max(maxSum, currentSum);
        }

        return maxSum;
    }
};
```



5. Search a 2D Matrix ||

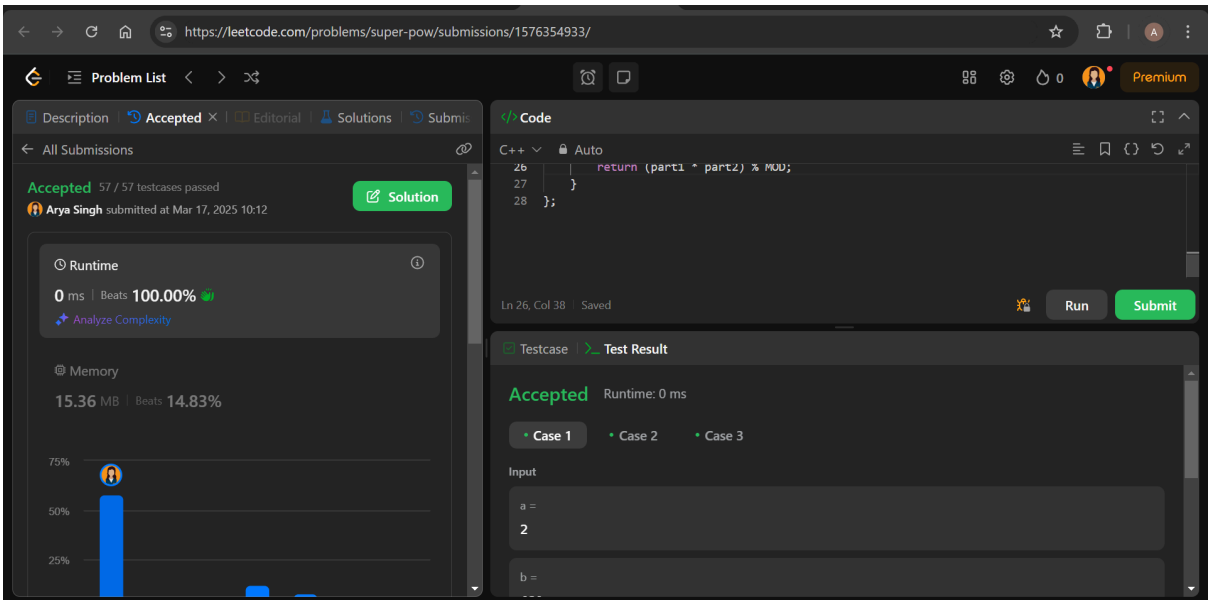
```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size(), cols = matrix[0].size();
        int row = 0, col = cols - 1; // Start from top-right

        while (row < rows && 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

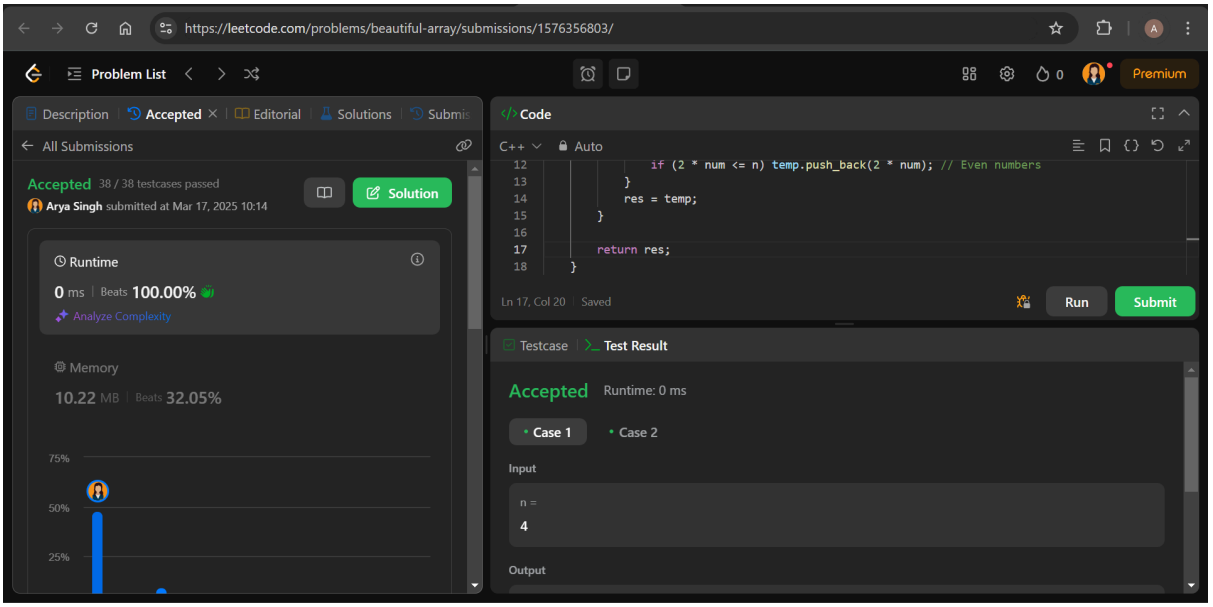
```
class Solution {
public:
    const int MOD = 1337;

    // Modular exponentiation (a^k % 1337)
    int modPow(int a, int k) {
        a %= MOD;
        int result = 1;
        for (int i = 0; i < k; i++) {
            result = (result * a) % MOD;
        }
        return result;
    }

    // Recursive function to compute (a^b) % 1337
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1; // Base case: a^0 = 1

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

        // Compute (a^(b/10))^10 and (a^lastDigit)
        int part1 = modPow(superPow(a, b), 10);
        int part2 = modPow(a, lastDigit);

        return (part1 * part2) % MOD;
```

```
    }
};
```



7. Beautiful Array

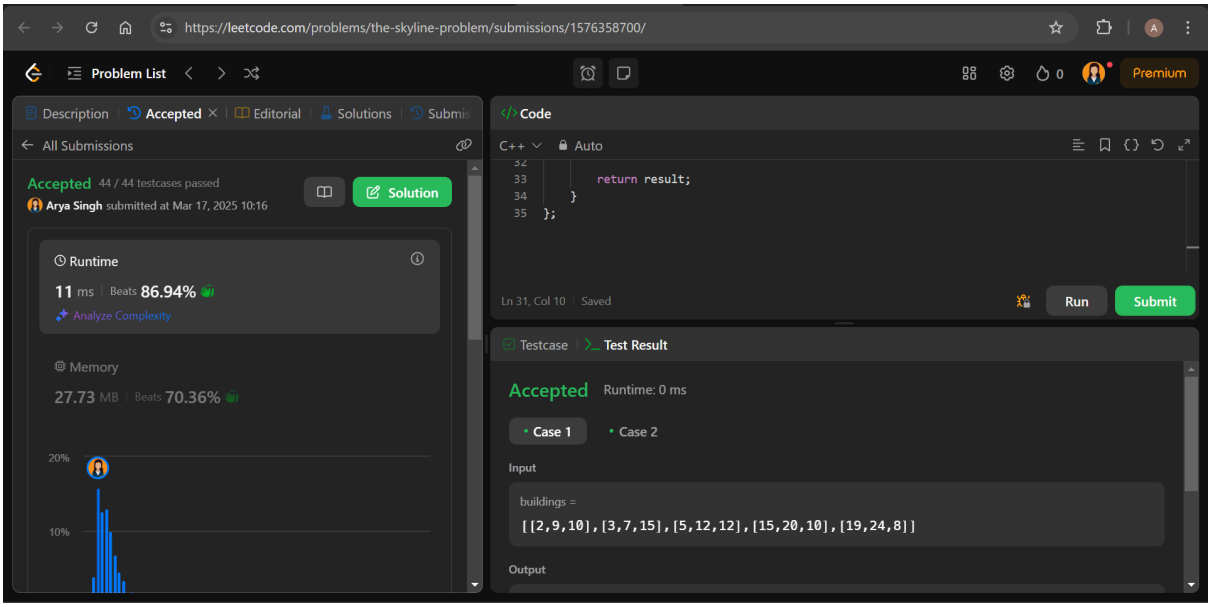
```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> res = {1};

        while (res.size() < n) {
            vector<int> temp;
            for (int num : res) {
                if (2 * num - 1 <= n) temp.push_back(2 * num - 1); // Odd numbers
            }
            for (int num : res) {
                if (2 * num <= n) temp.push_back(2 * num); // Even numbers
            }
            res = temp;
        }

        return res;
    }
};
```



8. The Skyline Problem

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
            for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Building start (negative height)
            events.emplace_back(b[1], b[2]);  // Building end (positive height)
        }

        sort(events.begin(), events.end());

        multiset<int> heights = {0};
        vector<vector<int>> result;
        int prevMax = 0;

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

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

            int currMax = *heights.rbegin();
            if (currMax != prevMax) {
                result.push_back({x, currMax});
                prevMax = currMax;
            }
```

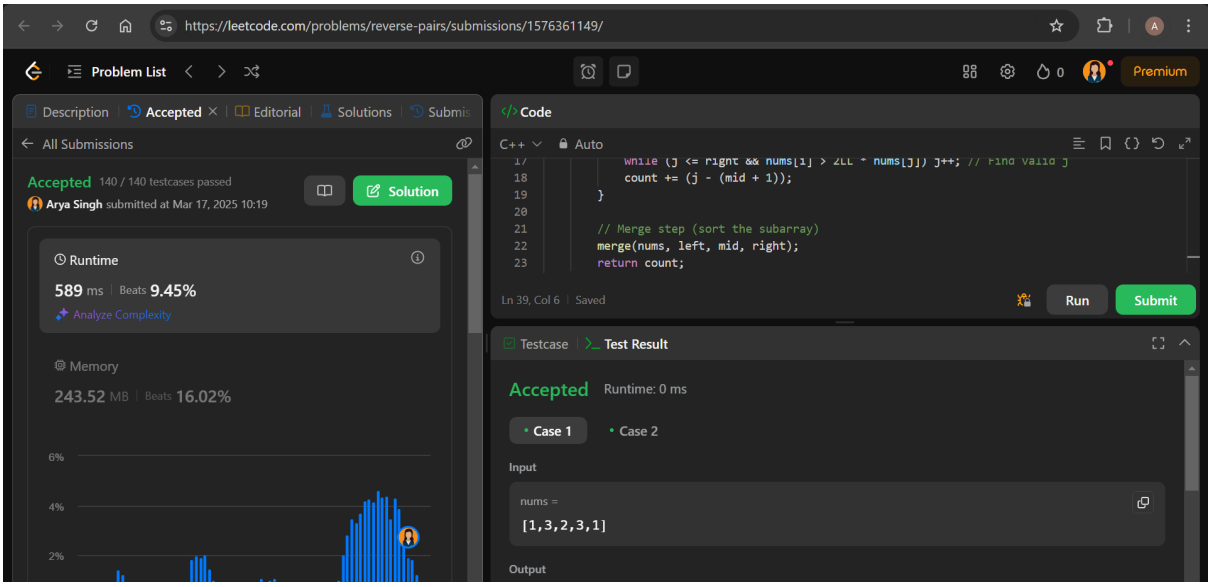
```
        }

        return result;
    }
};
```



9. Reverse Pairs

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

```
        // Merge step (sort the subarray)
        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];
    }
};
```

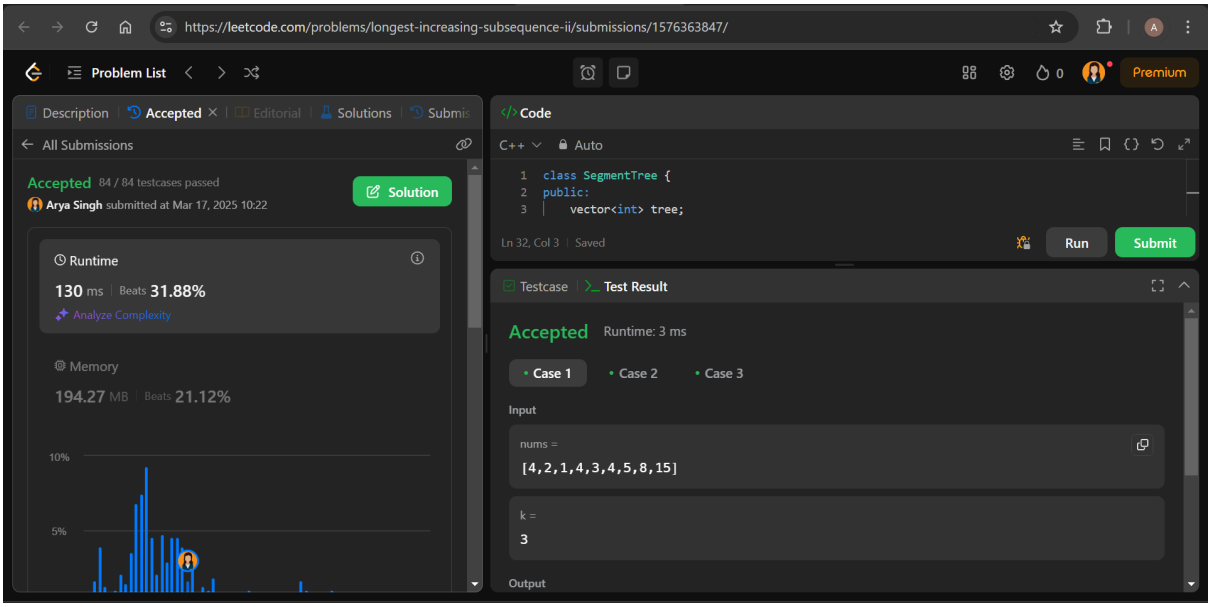


10. Longest Increasing Subsequence ||

```
class SegmentTree {
public:
    vector<int> tree;
    int size;
```

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

    void update(int index, int value, int node = 1, int start = 1, int end = 100000)
{
        if (start == end) {
            tree[node] = max(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 = 1, int end = 100000) {
        if (left > right) return 0;  // No valid range
        if (start > right || end < left) return 0;
        if (start >= left && 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) {
        SegmentTree segTree(100000);
        int maxLIS = 1;

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

        return maxLIS;
    }
```

};


https://leetcode.com/problems/longest-increasing-subsequence-ii/submissions/1576363847/
Problem List
Premium
Description
Accepted
Editorial
Solutions
All Submissions
Accepted 84 / 84 testcases passed
Arya Singh submitted at Mar 17, 2025 10:22
Solution
Runtime
130 ms | Beats 31.88%
Analyze Complexity
Memory
194.27 MB | Beats 21.12%
10%
5%
Code
C++
Auto
1 class SegmentTree {
2 public:
3 vector<int> tree;
Ln 32, Col 3 | Saved
Run
Submit
Testcase
Test Result
Accepted
Runtime: 3 ms
Case 1
Case 2
Case 3
Input
nums =
[4,2,1,4,3,4,5,8,15]
k =
3
Output