Name-Jasmine Kaur
UID-22BCS11841
Section-22BCS_IOT-607-B

Assignment-04

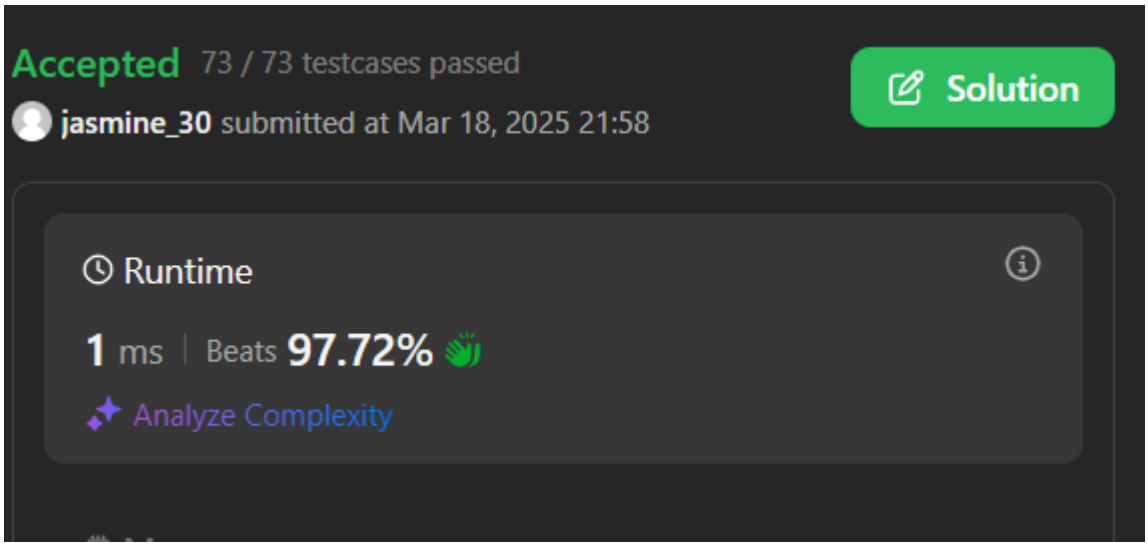
Advanced Programming Lab - 2 (22CSP-351)

Divide and Conquer

Question 1: Longest Nice Substring

Code:

```
class Solution { public:     string
longestNiceSubstring(string s) {         int
n = s.length();
        if (n < 2) return "";

        unordered_set<char> st(s.begin(), s.end());

        for (int i = 0; i < n; i++) {
            if (st.count(tolower(s[i])) && st.count(toupper(s[i]))) continue;
string left = longestNiceSubstring(s.substr(0, i));            string right =
longestNiceSubstring(s.substr(i + 1));            return left.length() >=
right.length() ? left : right;
        }
return s;
    }
};
```

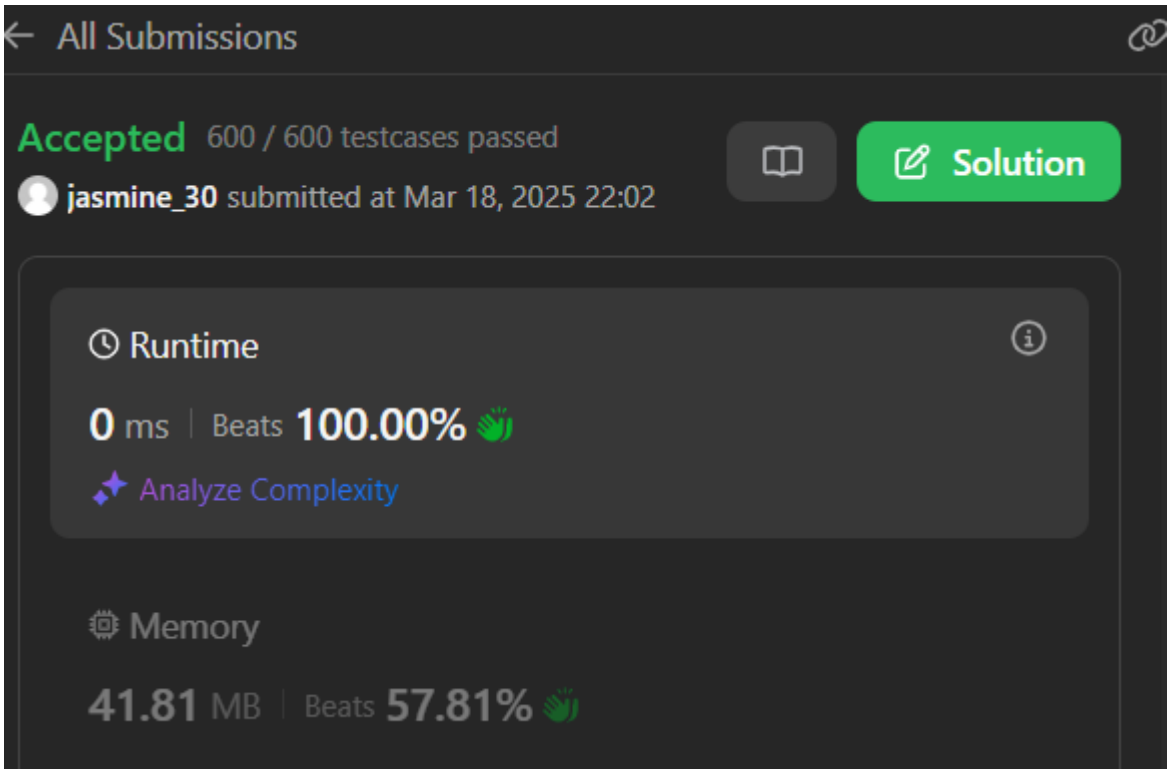


Question 2: Reverse Bits

Code:

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

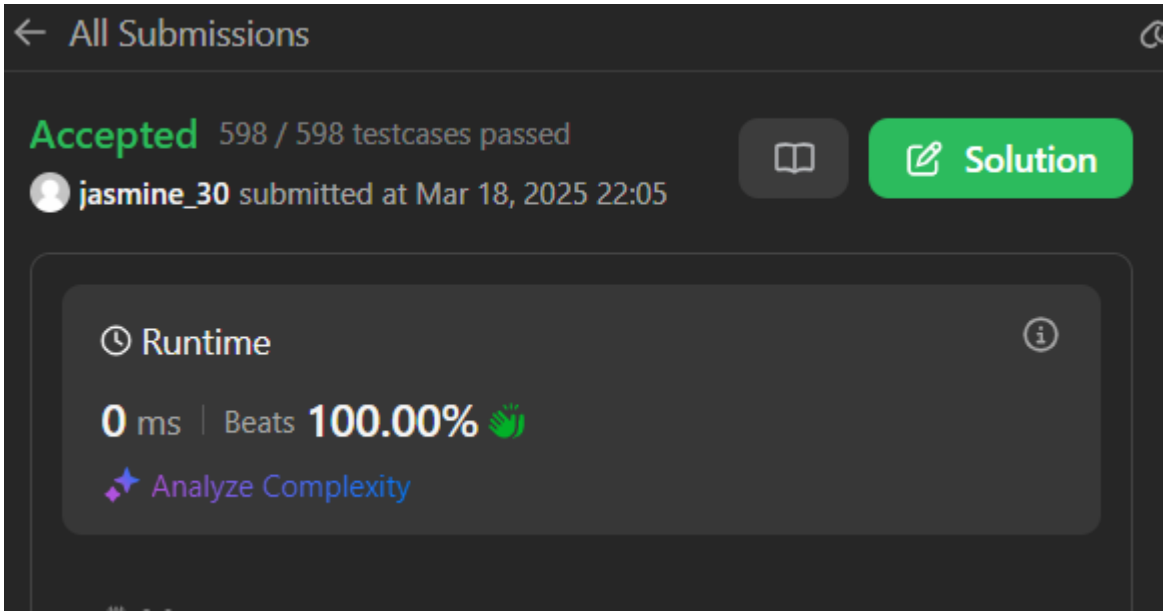
};



Question 3: Number of 1 Bits

Code:

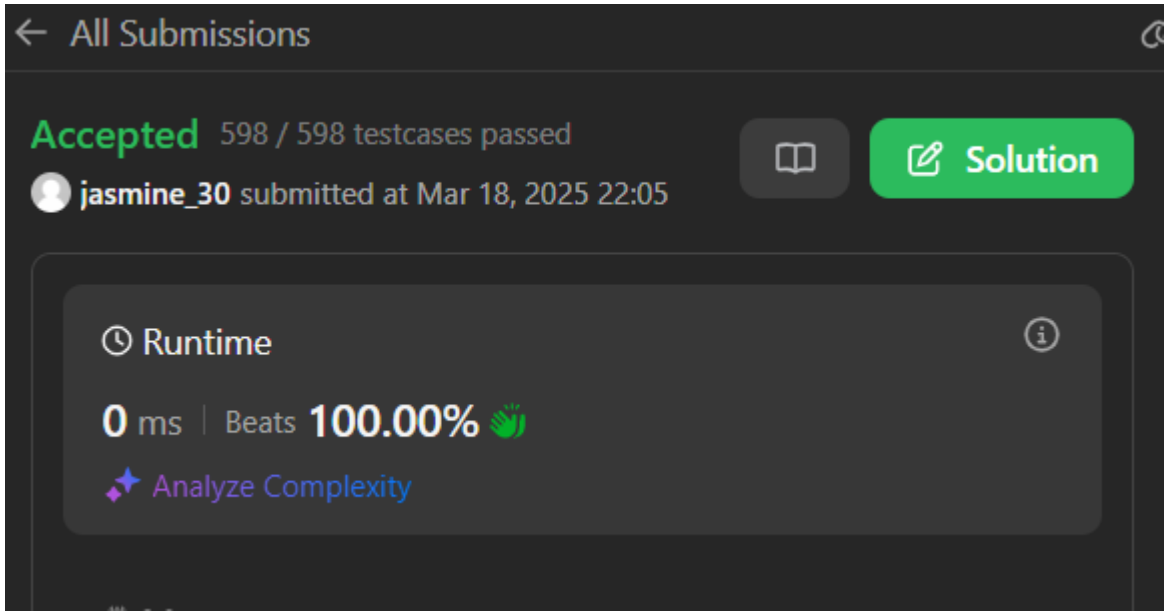
class Solution { public: int
hammingWeight(int n) {
int count = 0; while (n) {
count += (n & 1); n
>>= 1;
} return
count;
}
};



Question 4: Maximum Subarray

Code:

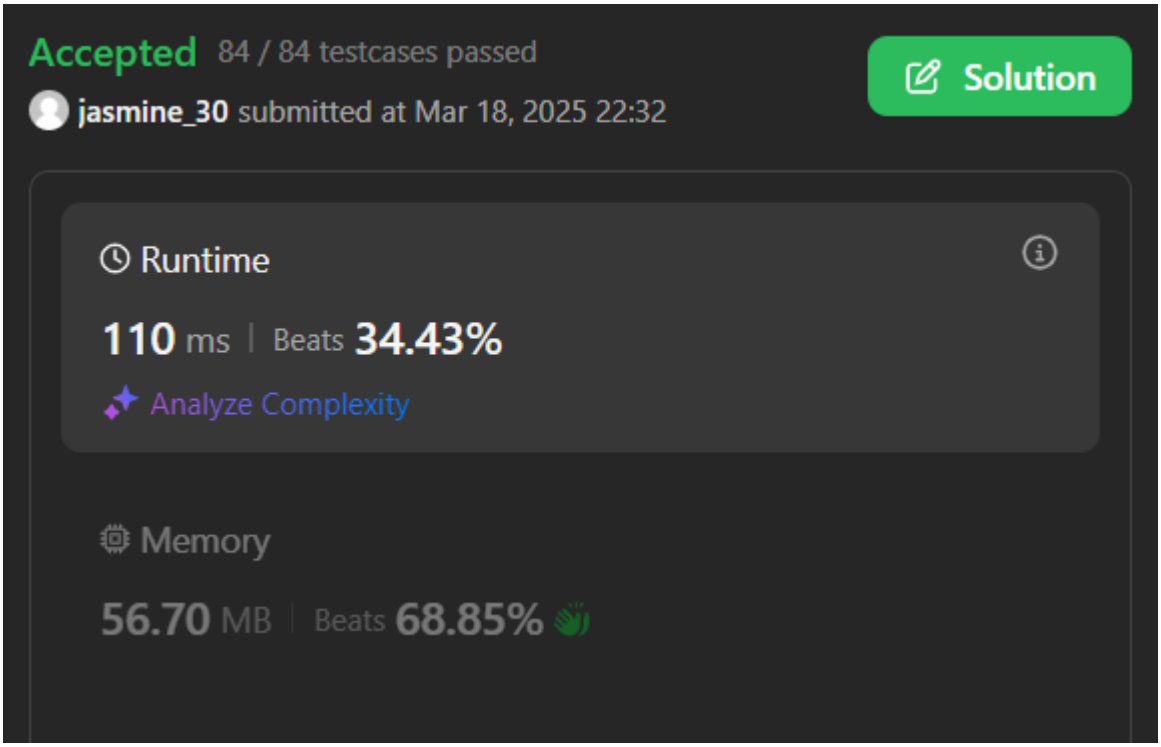
```
class Solution { public:
    int maxSubArray(vector<int>& nums) {
int maxSum = nums[0], curSum = 0;
for (int num : nums) {            curSum =
max(num, curSum + num);
            maxSum = max(maxSum, curSum);
        }
        return maxSum;
    }
};
```



Question 5: Search a 2D Matrix II

Code:

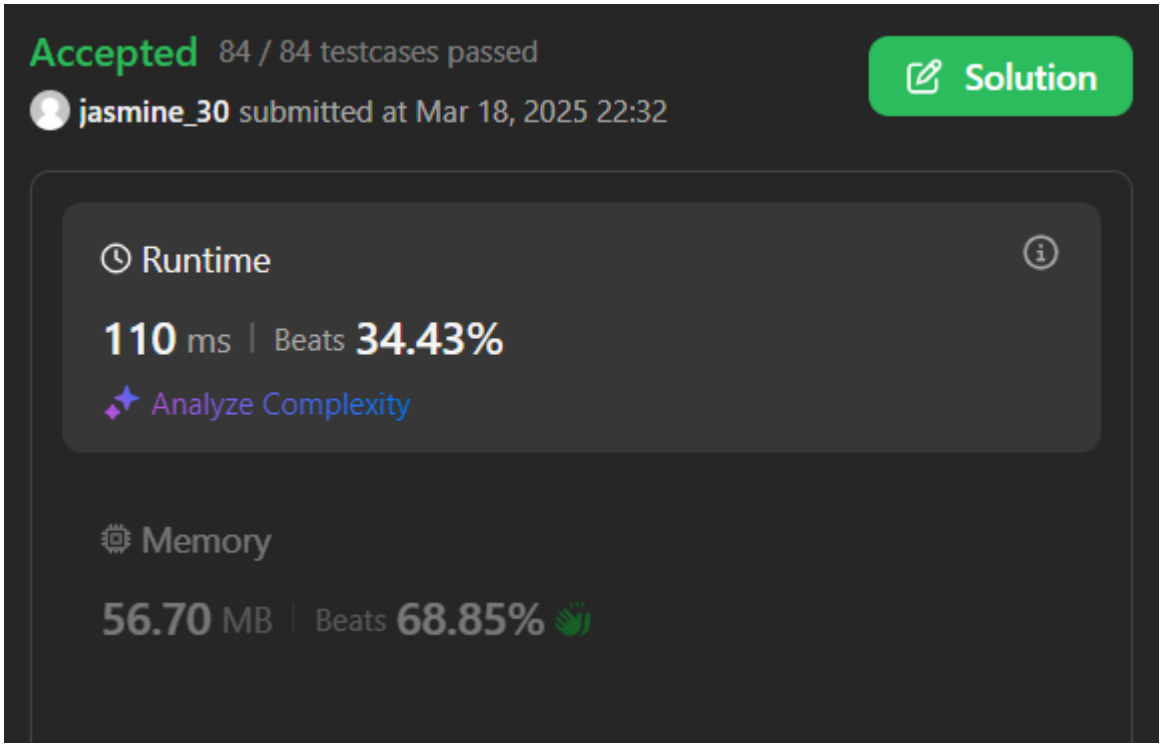
```
class Solution { public:     bool
searchMatrix(vector<vector<int>>& matrix, int target) {
int rows = matrix.size(), cols = matrix[0].size();         int r = 0, c
= cols - 1;

        while (r < rows && c >= 0) {            if
(matrix[r][c] == target) return true;
else if (matrix[r][c] > target) c--;            else
r++;
        }

        return false;
    }
};
```



Question 6: Super Pow

Code:

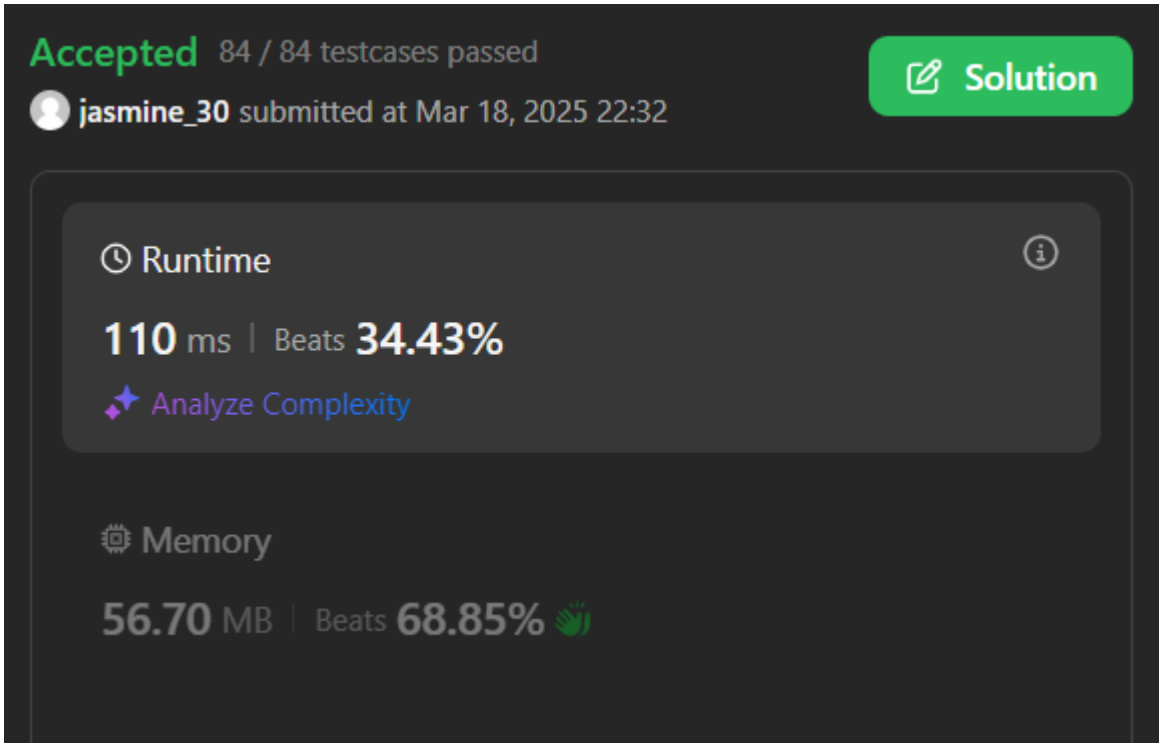
```
class Solution { public:
const int MOD = 1337;
int modPow(int x, int n) {
int res = 1;
        x %= MOD;
while (n) {
            if (n % 2) res = (res * x) % MOD;
x = (x * x) % MOD;            n /= 2;
}         return res;
    }
    int superPow(int a, vector<int>& b) {
int res = 1;
        for (int digit : b) {
            res = modPow(res, 10) * modPow(a, digit) % MOD;
        }
return res;
    }
};
```



Question 7: Beautiful Array

Code:

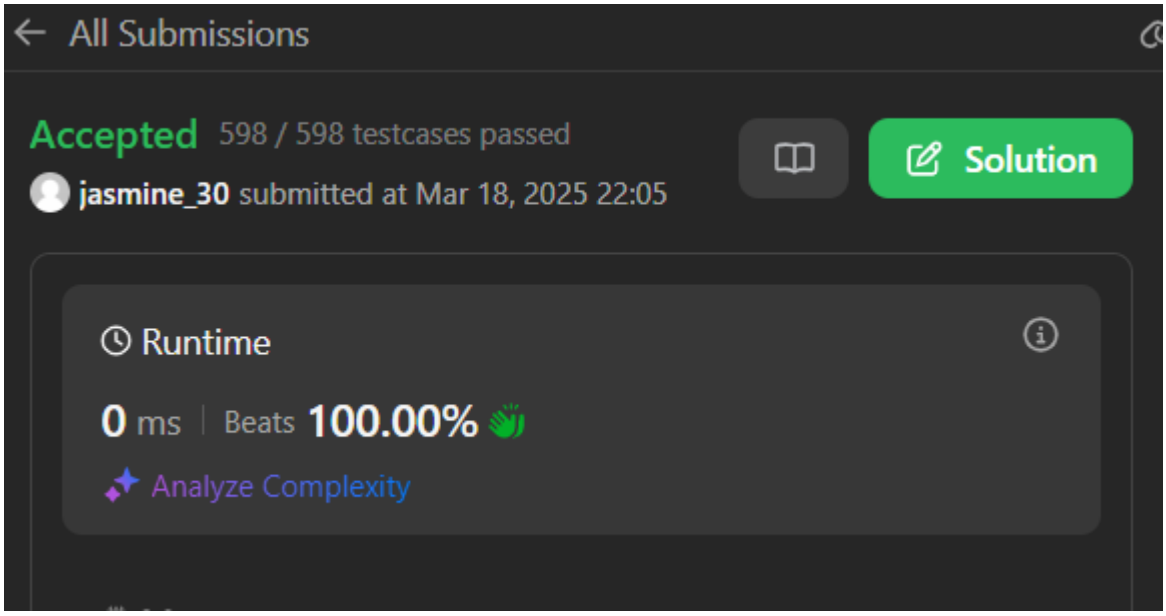
```
class Solution { public:
vector<int> beautifulArray(int n) {
vector<int> res = {1};        while
(res.size() < n) {            vector<int>
temp;
            for (int num : res) if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1);
for (int num : res) if (num * 2 <= n) temp.push_back(num * 2);            res =
temp;
        }
return res;
    }
};
```



Question 8: The Skyline Problem

Code:

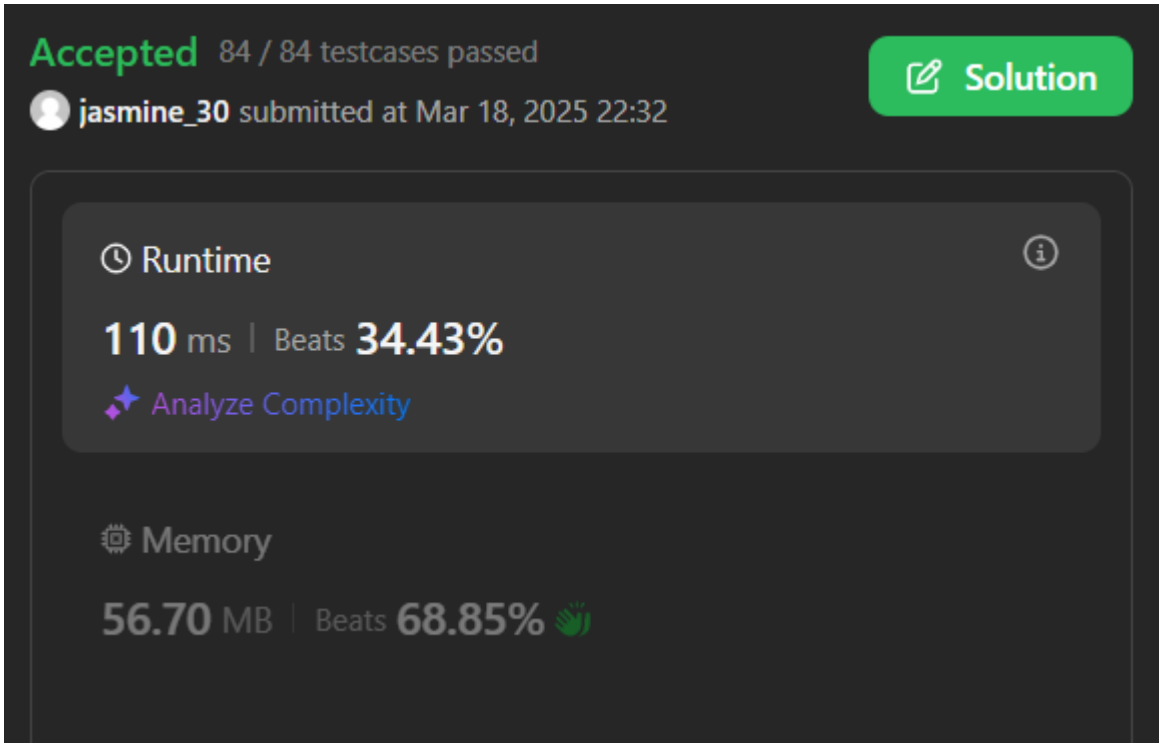
```
class Solution { public:     vector<vector<int>>
getSkyline(vector<vector<int>>& buildings) {         vector<pair<int,
int>> events;         for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Start of building
            events.emplace_back(b[1], b[2]);  // End of building
        }
        sort(events.begin(), events.end());

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

        for (auto& [x, h] : events) {
            if (h < 0) heights.insert(-h); // Insert height for start
else heights.erase(heights.find(h)); // Remove height for end
int curMax = *heights.rbegin();             if (curMax != prevMax) {
res.push_back({x, curMax});
                prevMax = curMax;
            }
}         return
res;
    }
};
```



Question 9: Reverse Pairs

Code:

```
class Solution { public:     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));
        }

        vector<int> sorted;         int i =
left, k = mid + 1;         while (i <= mid
&& k <= right) {
            if (nums[i] <= nums[k]) sorted.push_back(nums[i++]);
else sorted.push_back(nums[k++]);
        }
        while (i <= mid) sorted.push_back(nums[i++]);
while (k <= right) sorted.push_back(nums[k++]);

        for (int i = left; i <= right; i++) nums[i] = sorted[i - left];

        return count;
    }

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



Question 10: Longest Increasing Subsequence II

Code:

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

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

       void update(int index, int value, int node, int start, int end) {
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, int start, int end) {
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));
       }

       void update(int index, int value) {
           update(index, value, 1, 1, size);
       }

       int query(int left, int right) {
```

```
        return query(left, right, 1, 1, size);
    }
};

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

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

    return maxLength;
}
};
```