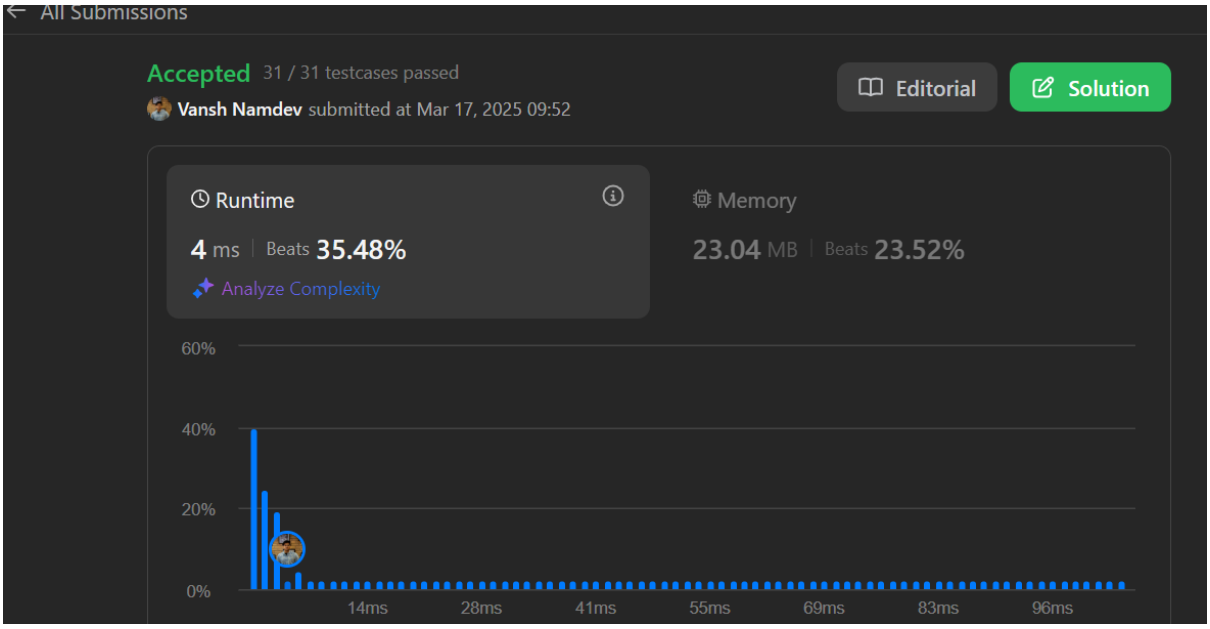
108. Convert Sorted Array to Binary Search Tree

- **Solution Code:**

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildBST(nums,0,nums.size()-1);
    }
    TreeNode* buildBST(vector<int>& nums,int left,int right){
        if(left>right){
            return nullptr;
        }
        int mid=left+(right-left)/2;
        TreeNode* node=new TreeNode(nums[mid]);
        node->left=buildBST(nums,left,mid-1);
        node->right=buildBST(nums,mid+1,right);
        return node;
    }
};
```

- **Screenshot:**

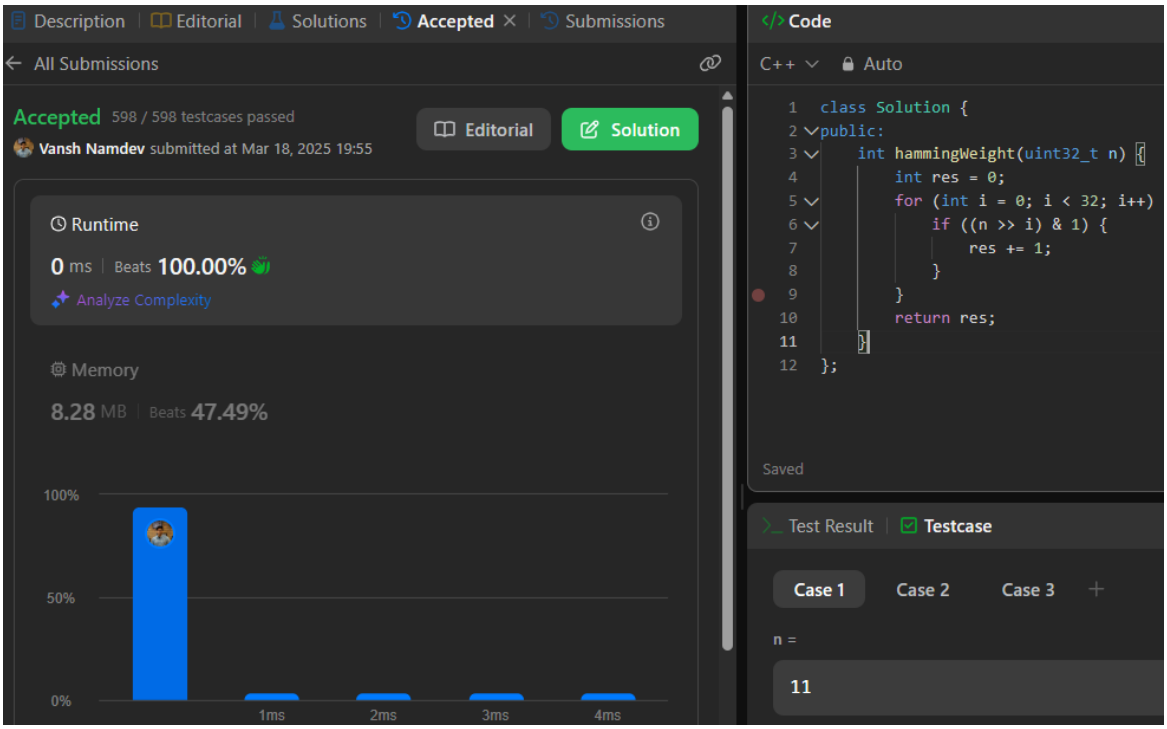


191. Number of 1 Bits

- **Source Code:**

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

- **Screenshot:**



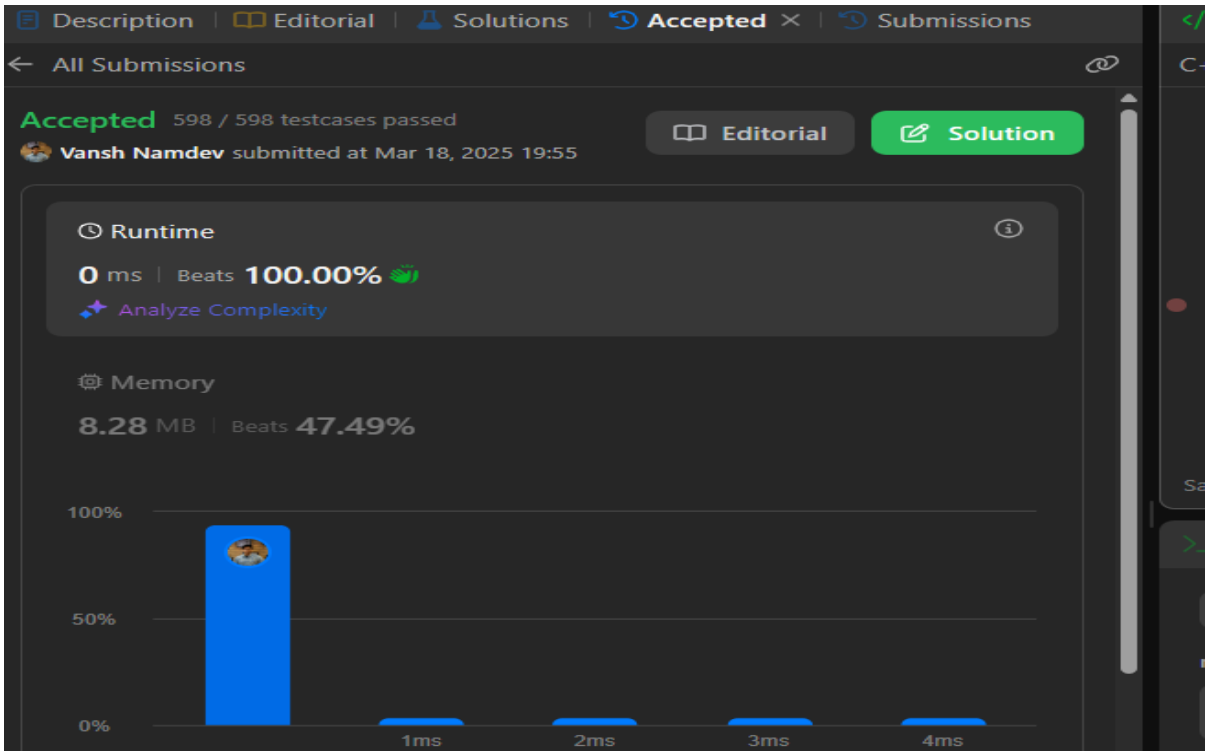
912. Sort an Array

- **Source Code:**

```
class Solution {
public:

    void merge(std::vector<int>& array, int left, int mid, int
right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        vector<int> L(n1);
        vector<int> R(n2);

        for (int i = 0; i < n1; ++i)
            L[i] = array[left + i];
        for (int j = 0; j < n2; ++j)
            R[j] = array[mid + 1 + j];


        int i = 0;
        int j = 0;
        int k = left;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                array[k] = L[i];
                ++i;
            } else {
                array[k] = R[j];
                ++j;
            }
            ++k;
        }

        while (i < n1) {
            array[k] = L[i];
            ++i;
            ++k;
        }

        while (j < n2) {
            array[k] = R[j];
            ++j;
            ++k;
        }

    }
```

```
    void mergeSort(std::vector<int>& array, int left, int
right) {
        if (left >= right)
            return;

        int mid = left + (right - left) / 2;

        mergeSort(array, left, mid);
        mergeSort(array, mid + 1, right);

        merge(array, left, mid, right);
    }


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

        return nums;
    }
};
```

- **Screenshot:**



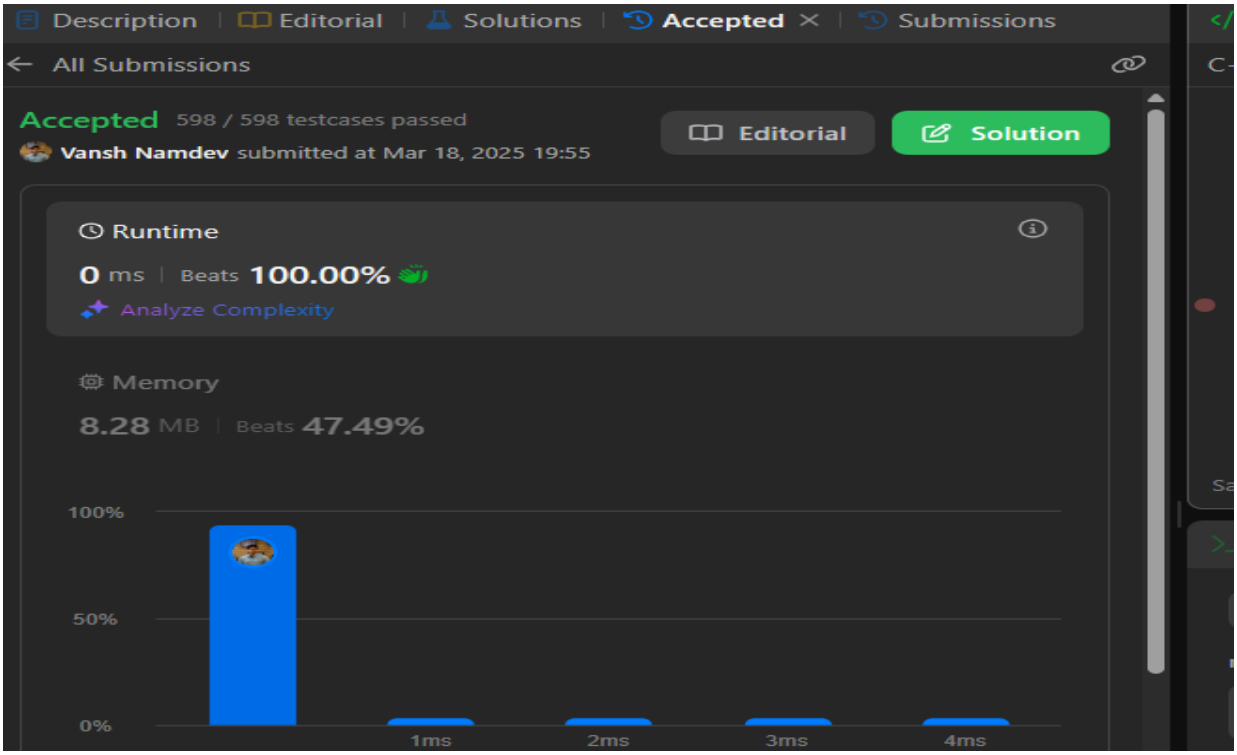
-

53. Maximum Subarray

- **Source Code:**

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

        for (int i = 1; i < nums.size(); i++) {
            maxEnding = max(maxEnding + nums[i], nums[i]);
            res = max(res, maxEnding);
        }
        return res;
    }
};
```

- **Screenshot:**

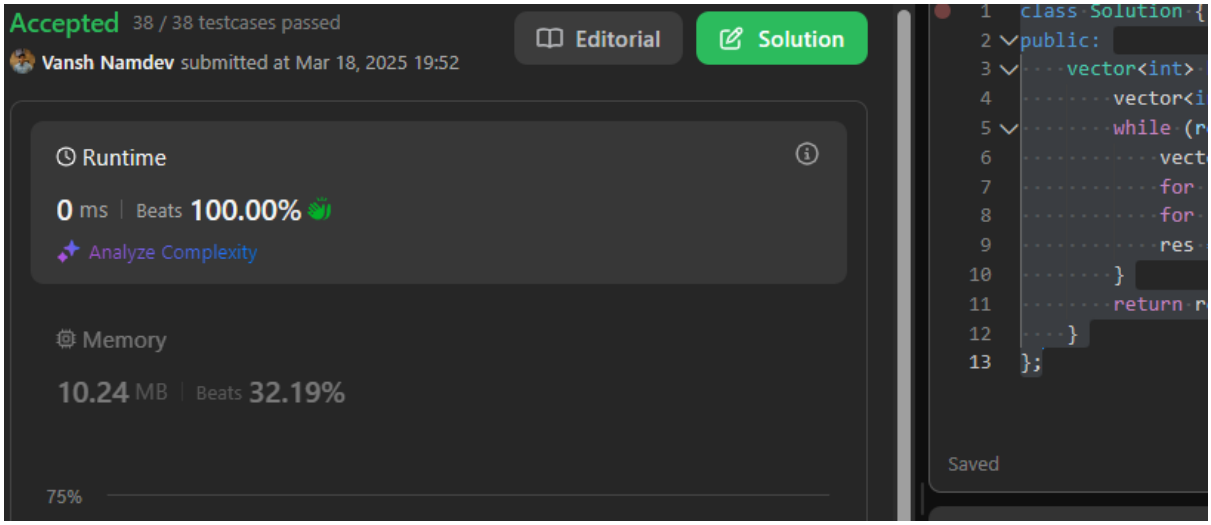


932. Beautiful Array

- **Source Code:**

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

- };**Screenshot:**
-

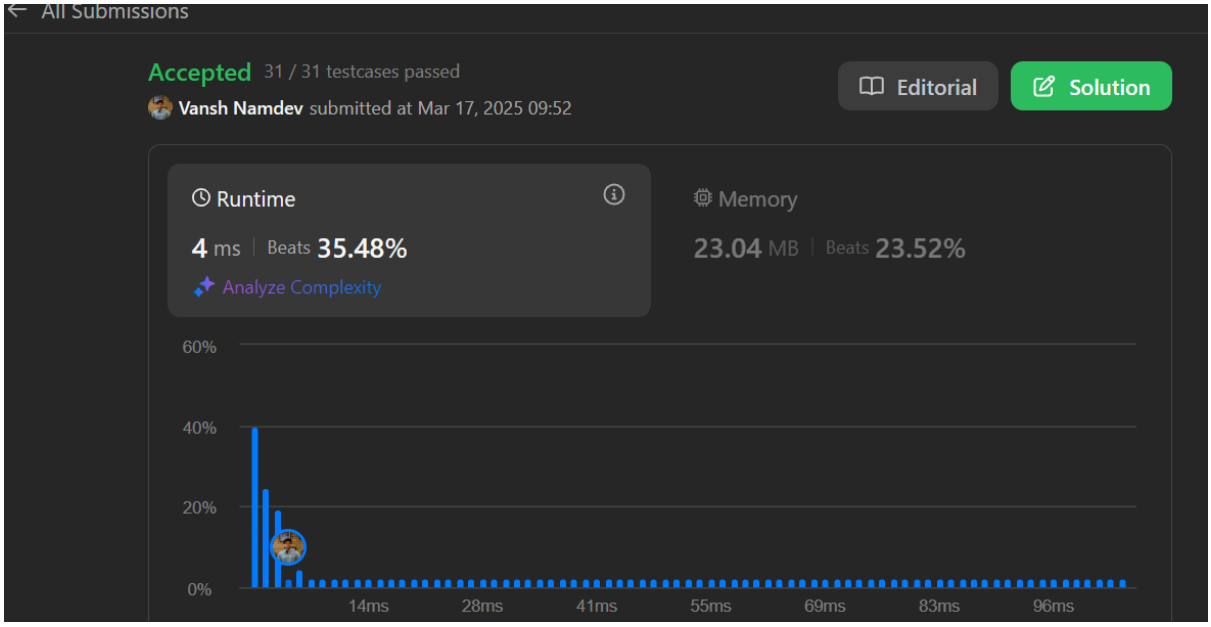


372. Super Pow

- **Source Code:**

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10
    {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)
            result = (result * a) % base;
        return result;
    }
public:
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a,
last_digit) % base;
    }
};
```

- **Screenshot:**

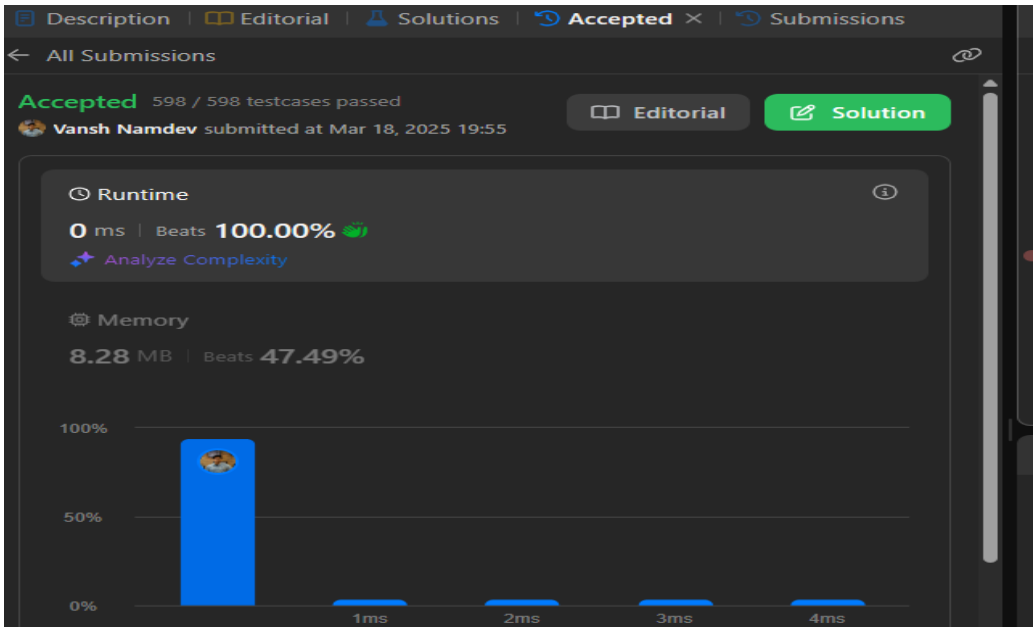


218. The Skyline Problem

- **Source Code:**

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        int edge_idx = 0;
        vector<pair<int, int>> edges;
        priority_queue<pair<int, int>> pq;
        vector<vector<int>> skyline;

        for (int i = 0; i < buildings.size(); ++i) {
            const auto &b = buildings[i];
            edges.emplace_back(b[0], i);
            edges.emplace_back(b[1], i);
        }

        std::sort(edges.begin(), edges.end());

        while (edge_idx < edges.size()) {
            int curr_height;
            const auto &[curr_x, _] = edges[edge_idx];
            while (edge_idx < edges.size() &&
                    curr_x == edges[edge_idx].first) {
                const auto &[_, building_idx] = edges[edge_idx];
                const auto &b = buildings[building_idx];
                if (b[0] == curr_x)
                    pq.emplace(b[2], b[1]);
                ++edge_idx;
            }
            while (!pq.empty() && pq.top().second <= curr_x)
                pq.pop();
            curr_height = pq.empty() ? 0 : pq.top().first;
            if (skyline.empty() || skyline.back()[1] != curr_height)
                skyline.push_back({curr_x, curr_height});
        }
        return skyline;
    }
};
```

- **Screenshot:**


Description
Editorial
Solutions
Accepted
Submissions
All Submissions
Accepted 598 / 598 testcases passed
Vansh Namdev submitted at Mar 18, 2025 19:55
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
8.28 MB | Beats 47.49%
100%
50%
0%
1ms
2ms
3ms
4ms