DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

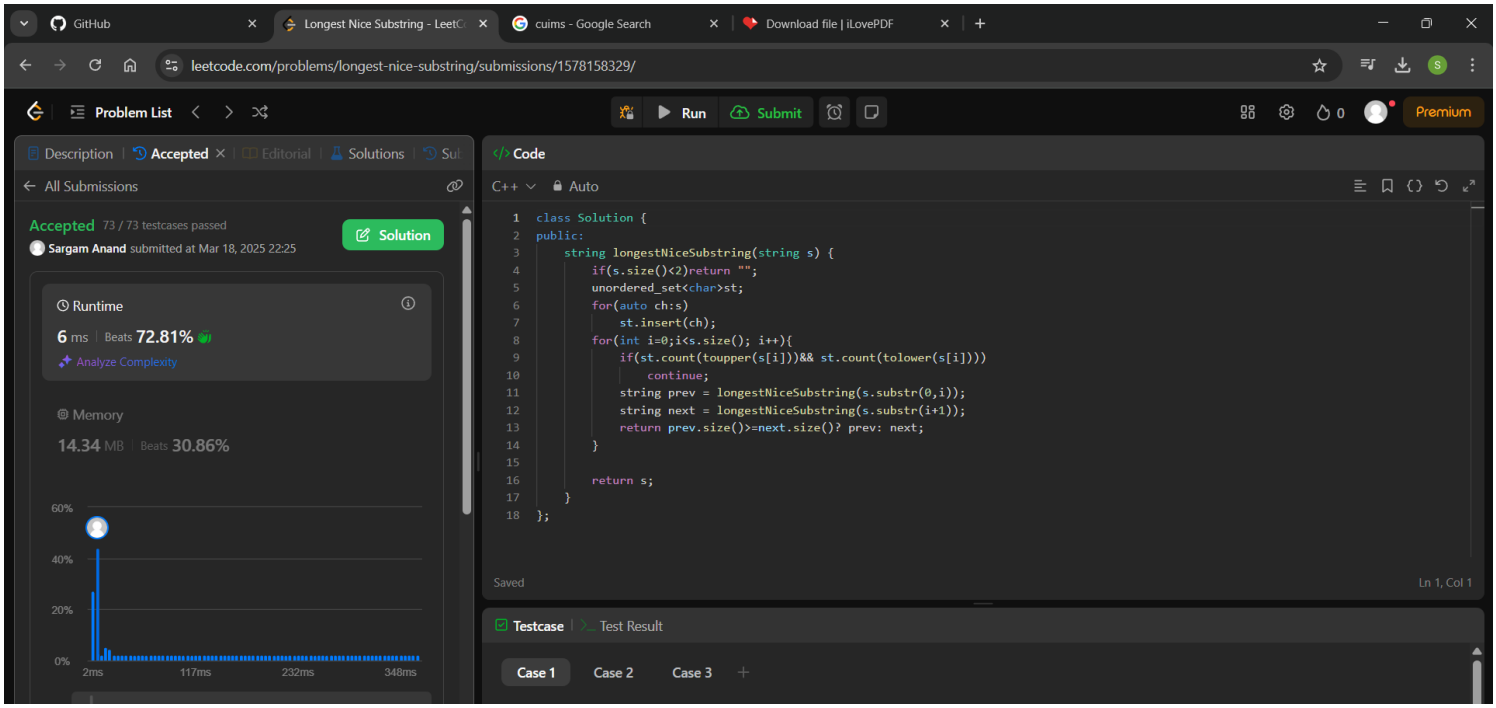
Assignment

Name: Sargam Anand
Branch: BE-CSE
Semester: 6th
Subject Name: Advanced Programming

UID: 22BCS14851
Section/Group: 607/B
Date of Performance: 05/03/25
Subject Code: 22CSP-351

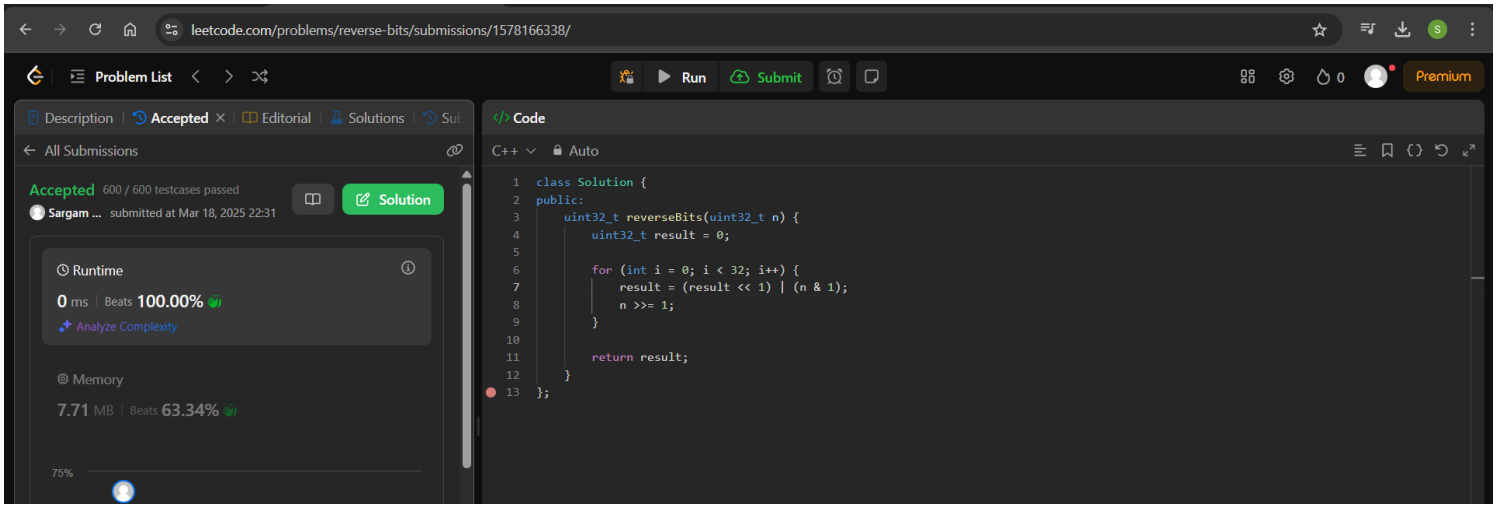
Code 1

```
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;
    }
```



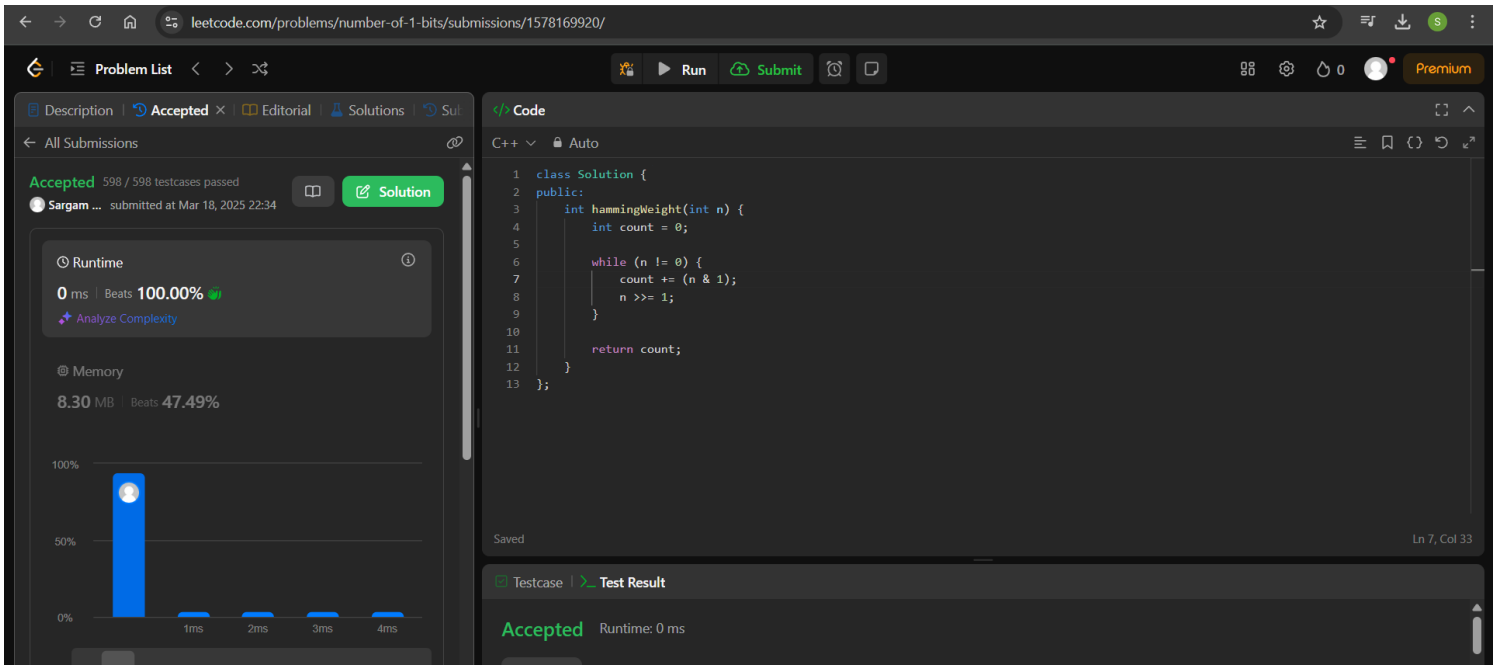
Code 2

```
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;
    }
};
```



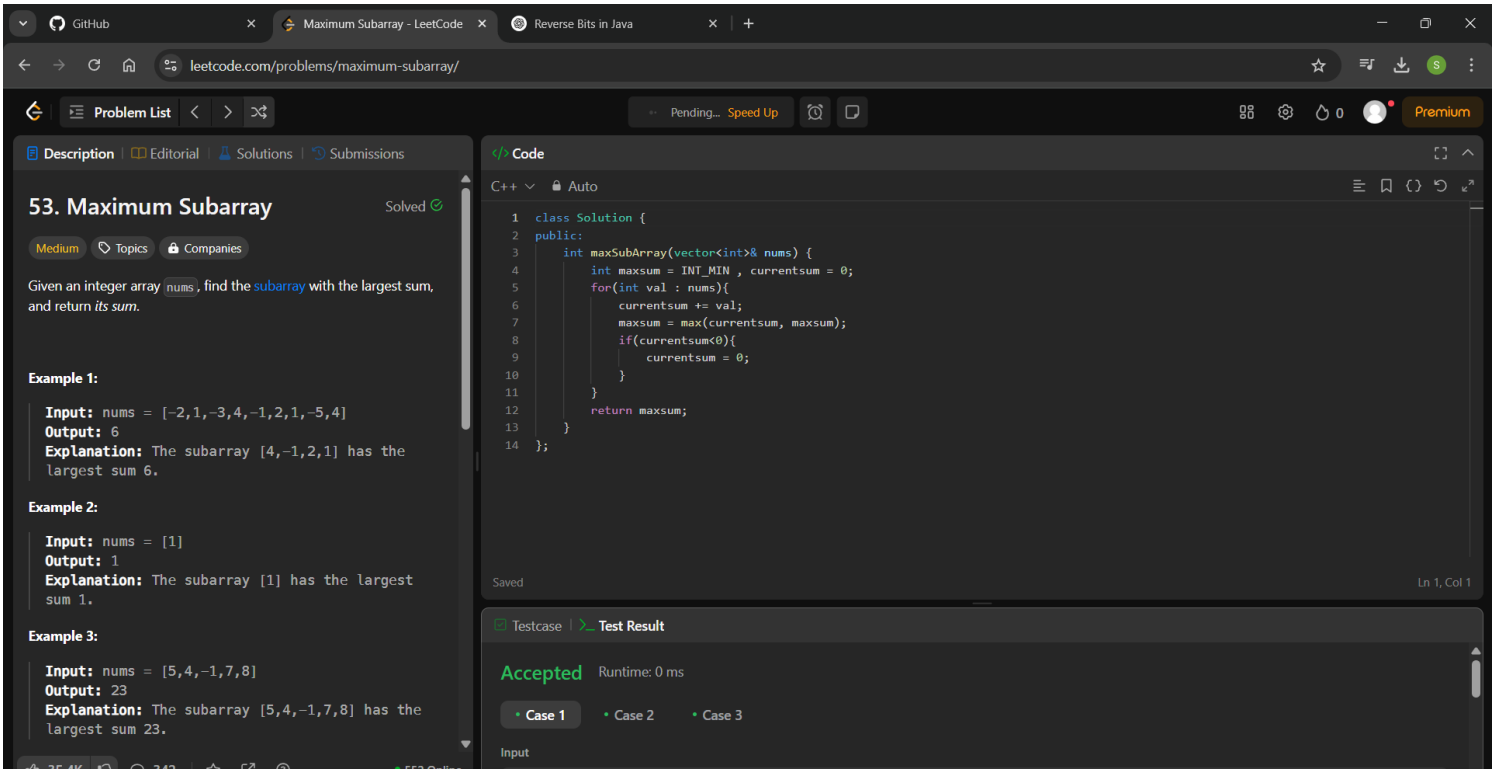
Code 3

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



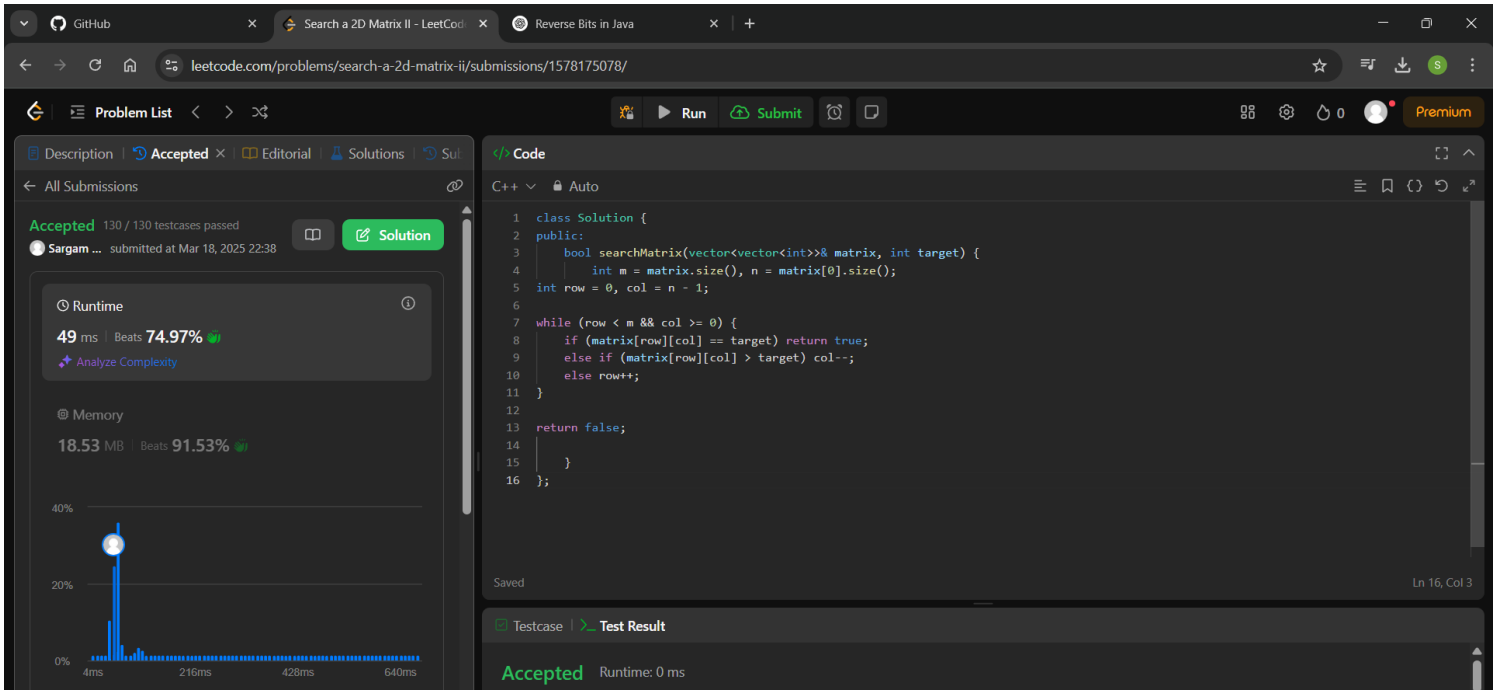
Code 4

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxsum = INT_MIN , currentsum = 0;
        for(int val : nums){
            currentsum += val;
            maxsum = max(currentsum, maxsum);
            if(currentsum<0){
                currentsum = 0;
            }
        }
        return maxsum;
    }
};
```



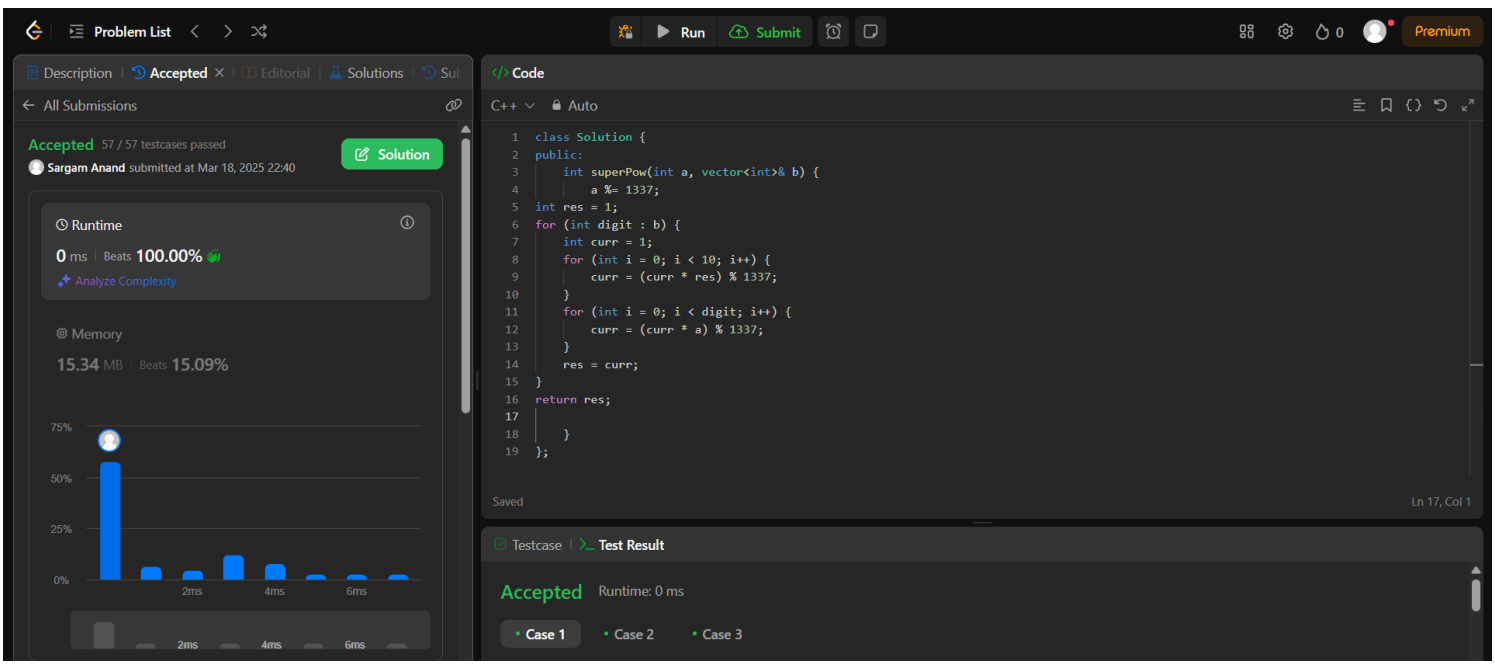
Code 5

```
int m = matrix.size(), n = matrix[0].size();
int row = 0, col = n - 1;

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

return false;
```



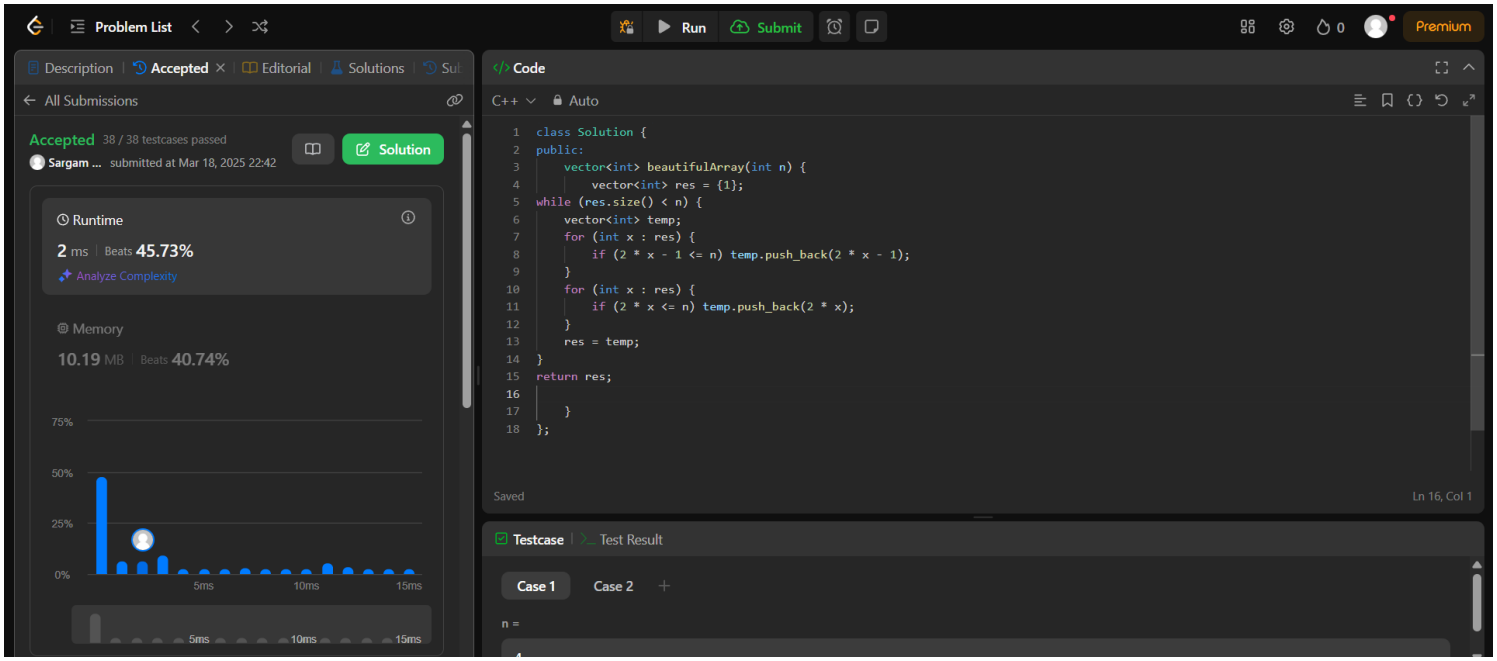
Code 6

```
a %= 1337;
int res = 1;
for (int digit : b) {
    int curr = 1;
    for (int i = 0; i < 10; i++) {
        curr = (curr * res) % 1337;
    }
    for (int i = 0; i < digit; i++) {
        curr = (curr * a) % 1337;
    }
    res = curr;
}
return res;
```



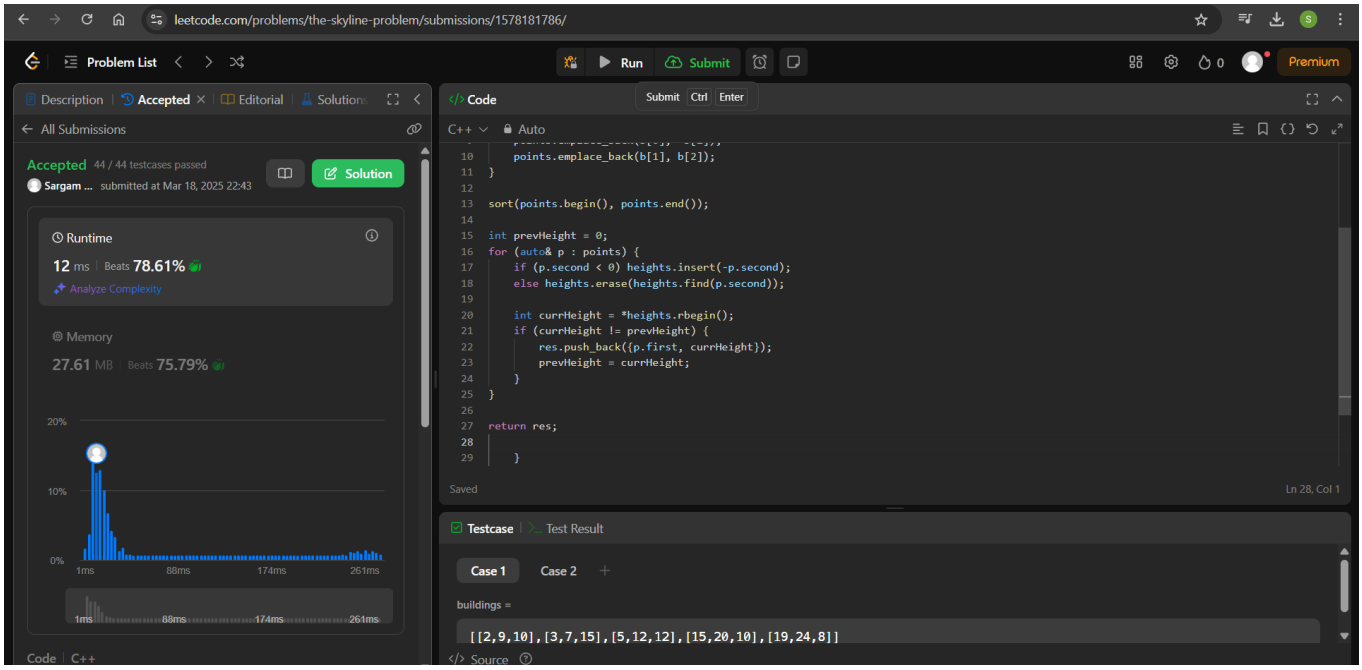
Code 7

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



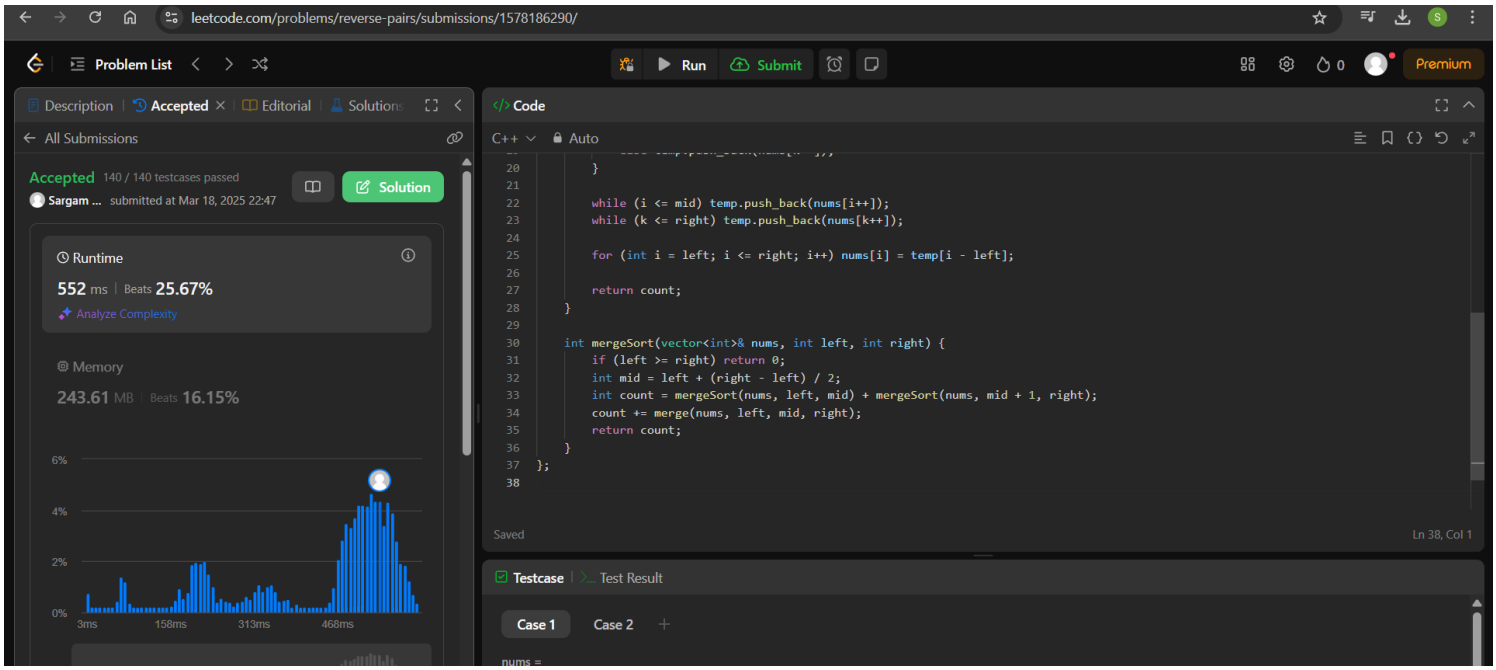
Code 8

```
vector<vector<int>> res;
multiset<int> heights = {0};
vector<pair<int, int>> points;

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

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

int prevHeight = 0;
for (auto& p : points) {
    if (p.second < 0) heights.insert(-p.second);
    else heights.erase(heights.find(p.second));

    int currHeight = *heights.rbegin();
    if (currHeight != prevHeight) {
        res.push_back({p.first, currHeight});
        prevHeight = currHeight;
    }
}

return res;
```



Code 9

```
int merge(vector<int>& nums, int left, int mid, int right) {
    int count = 0, 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> temp;
    int i = left, k = mid + 1;
    while (i <= mid && k <= right) {
        if (nums[i] <= nums[k]) temp.push_back(nums[i++]);
        else temp.push_back(nums[k++]);
    }

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

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

    return count;
}

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

return mergeSort(nums, 0, nums.size() - 1);
```



Code 10

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

private:
    int merge(vector<int>& nums, int left, int mid, int right) {
        int count = 0, 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> temp;
        int i = left, k = mid + 1;

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

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

        for (int i = left; i <= right; ++i) {
            nums[i] = temp[i - left];
        }

        return count;
    }

    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 += merge(nums, left, mid, right);

        return count;
    }
};
```