Assignment 4

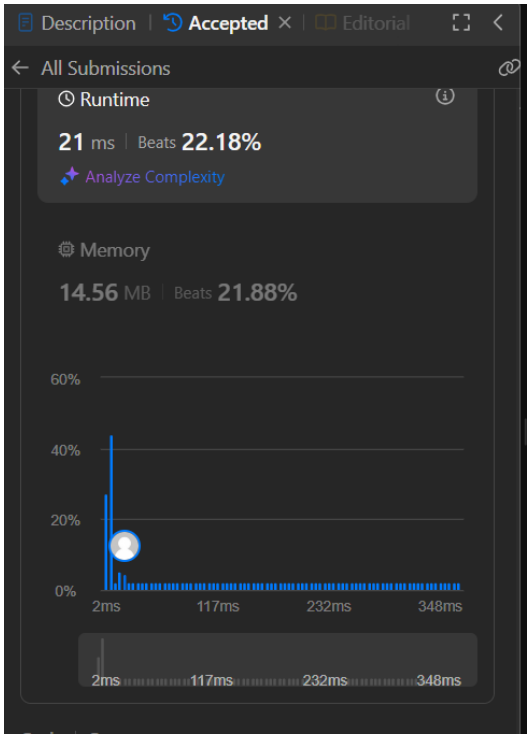
Name: Shivam Kumar Section : 606-B

UID: 22BCS10029

1. Longest Nice Substring:

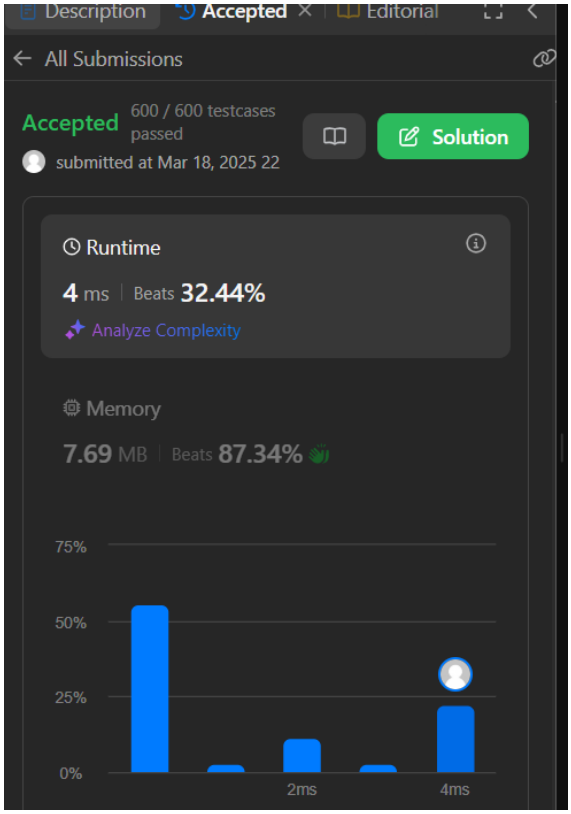
```
class Solution {
public:
    bool check(string s) {
        for (int i = 0; i < s.size(); i++) {
            char c = s[i];
            if (c <= 90) c += 32;
            else c -= 32;
            if (s.find(c) == string::npos) return false;
        }
        return true;
    }


    string longestNiceSubstring(string s) {
        string ans = "";
        for (int i = 0; i < s.size(); i++) {
            string res = "";
            res += s[i];
            for (int j = i + 1; j < s.size(); j++) {
                res += s[j];
                if (check(res) && res.size() > ans.size()) ans = res;
            }
        }
```

```
        return ans;
    }
};
```



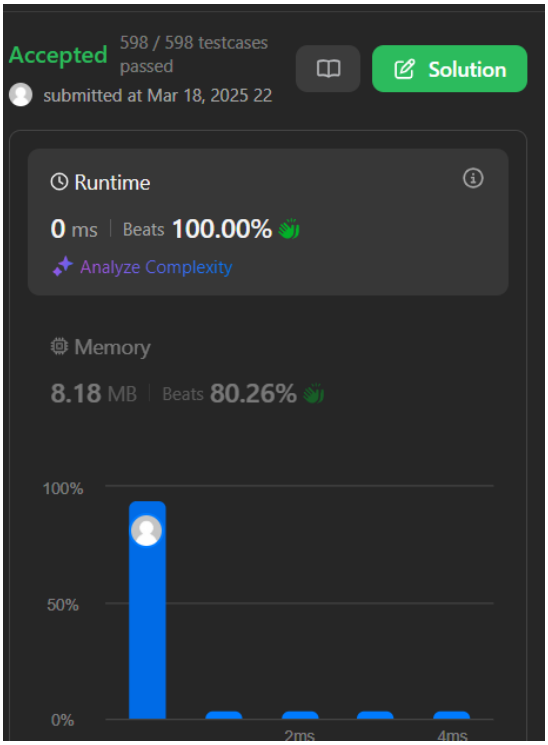
2. Reverse Bits:

```
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);
            n >>= 1;
        }
        return result;
    }
};
```



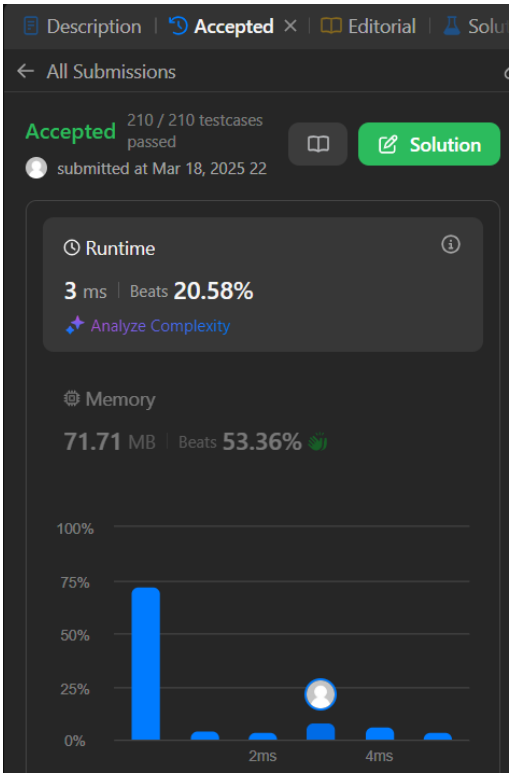
3. Number of 1 Bits:

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



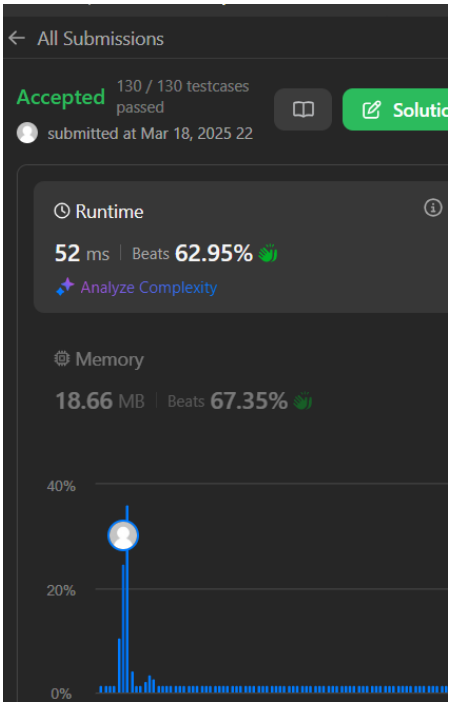
4. Maximum Subarray:

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

    for (int i = 1; i < nums.size(); ++i) {
        currentSum = max(nums[i], currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;

    }
};
```



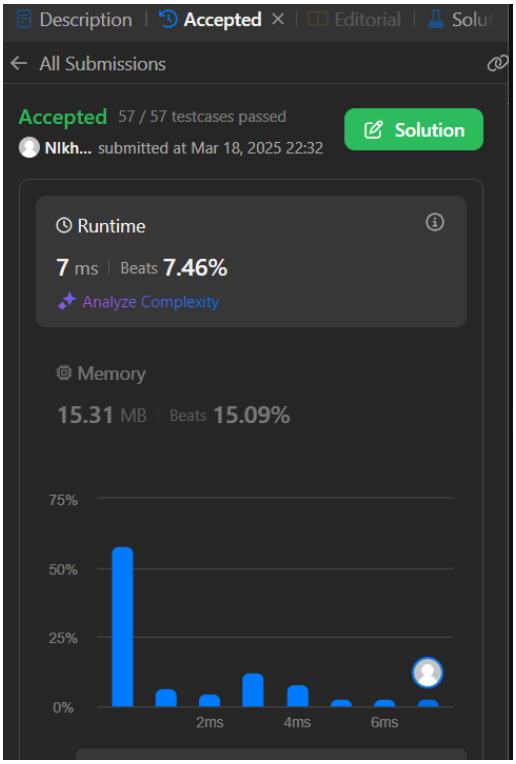
5. Search a 2D Matrix II:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        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;
    }
};
```



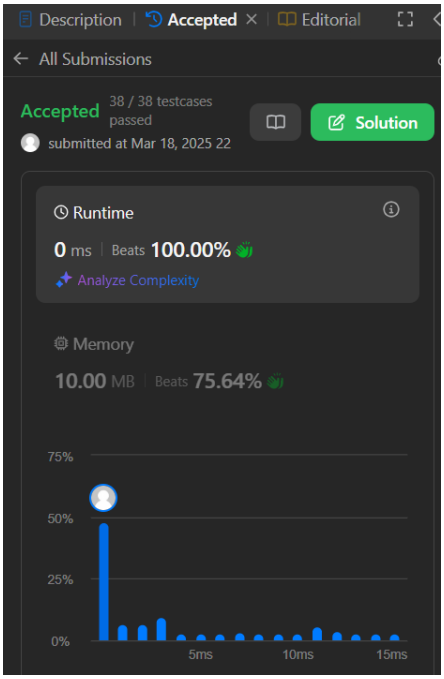
6. Super Pow

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

    int powerMod(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 = powerMod(result, 10) * powerMod(a, digit) % MOD;
        }
        return result;
    }
};
```



7. Beautiful Array:

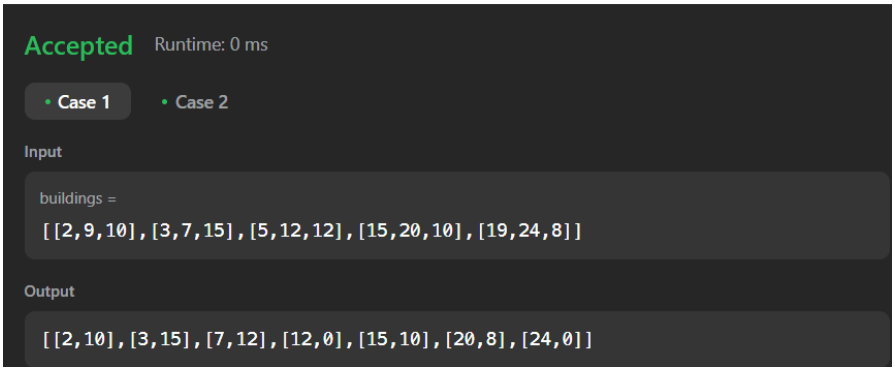
```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> result = {1};
        while (result.size() < n) {
            vector<int> temp;
            for (int x : result) {
                if (2 * x - 1 <= n) temp.push_back(2 * x - 1);
            }
            for (int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



8. The Skyline Problem:

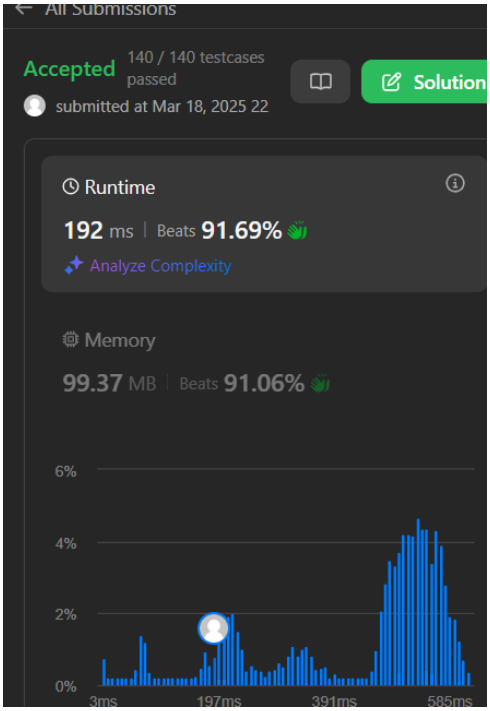
```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        vector<vector<int>> result;

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

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

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

        for (const auto& event : events) {
            int x = event.first, height = event.second;
            if (height < 0) {
                heights.insert(-height); // Add building height
            } else {
                heights.erase(heights.find(height)); // Remove building height
            }

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

```
        }

        return result;
    }
};
```



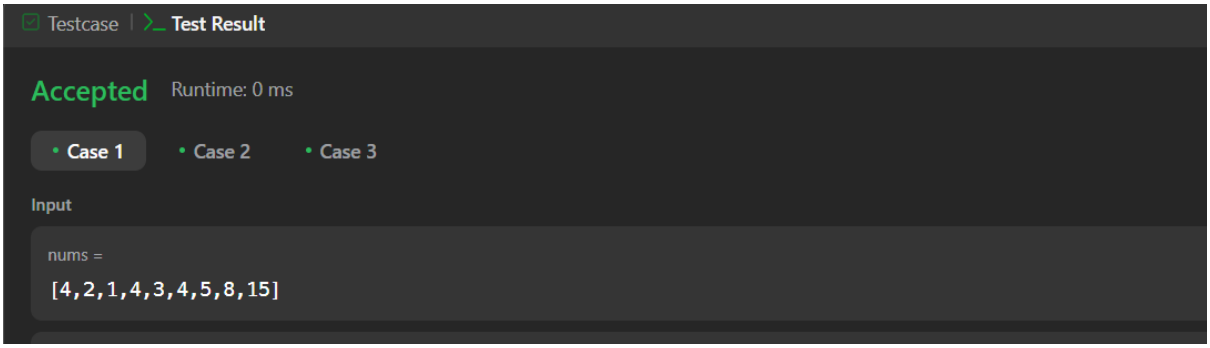
9. Reverse Pairs:

```
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);
        }

        inplace_merge(nums.begin() + left, nums.begin() + mid + 1, nums.begin() + right + 1);
        return count;
    }

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



10. Longest Increasing Subsequence II

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        int maxNum = *max_element(nums.begin(), nums.end());
        vector<int> dp(maxNum + 1, 0);

        for (int num : nums) {
            int left = max(0, num - k);
            int right = num - 1;
            int maxPrev = 0;

            for (int i = left; i <= right; i++) {
                maxPrev = max(maxPrev, dp[i]);
            }

            dp[num] = maxPrev + 1;
        }

        return *max_element(dp.begin(), dp.end());
    }
};
```


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[4,2,1,4,3,4,5,8,15]