Anirudh Gagneja

22BCS16527

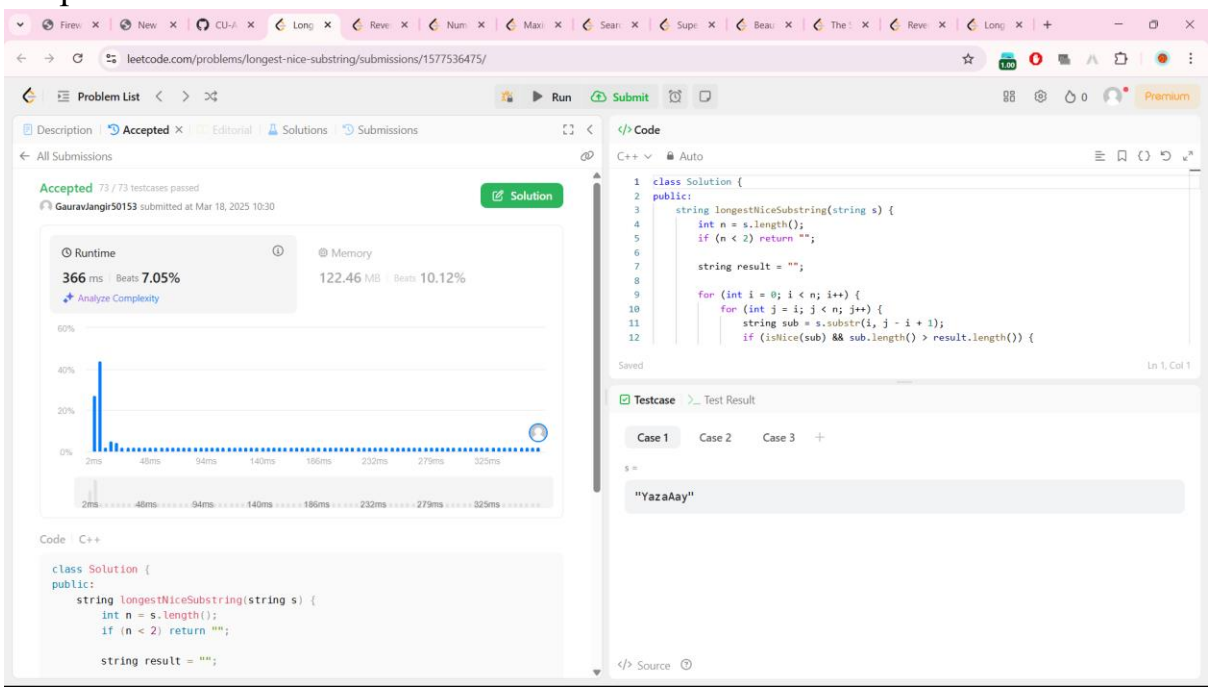
Assignment 04

Advanced Programming

1. **Longest Nice Substring:**

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

        string result = "";

        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                string sub = s.substr(i, j - i + 1);
                if (isNice(sub) && sub.length() > result.length()) {
                    result = sub;
                }
            }
        }

        return result;
    }

private:
    bool isNice(const string& str) {
        unordered_set<char> charSet(str.begin(), str.end());
        for (char c : str) {
            if (charSet.count(tolower(c)) == 0 || charSet.count(toupper(c)) == 0) {
                return false;
            }
        }
        return true;
    }
};
```

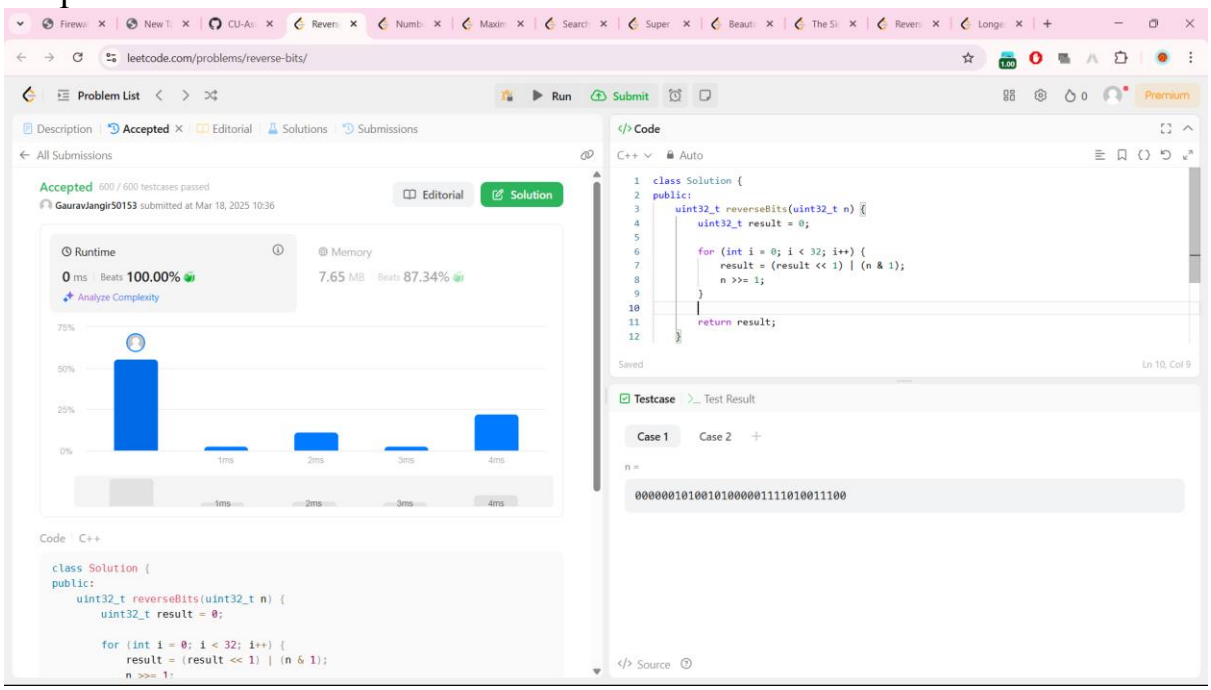
Output:



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;
    }
};
```

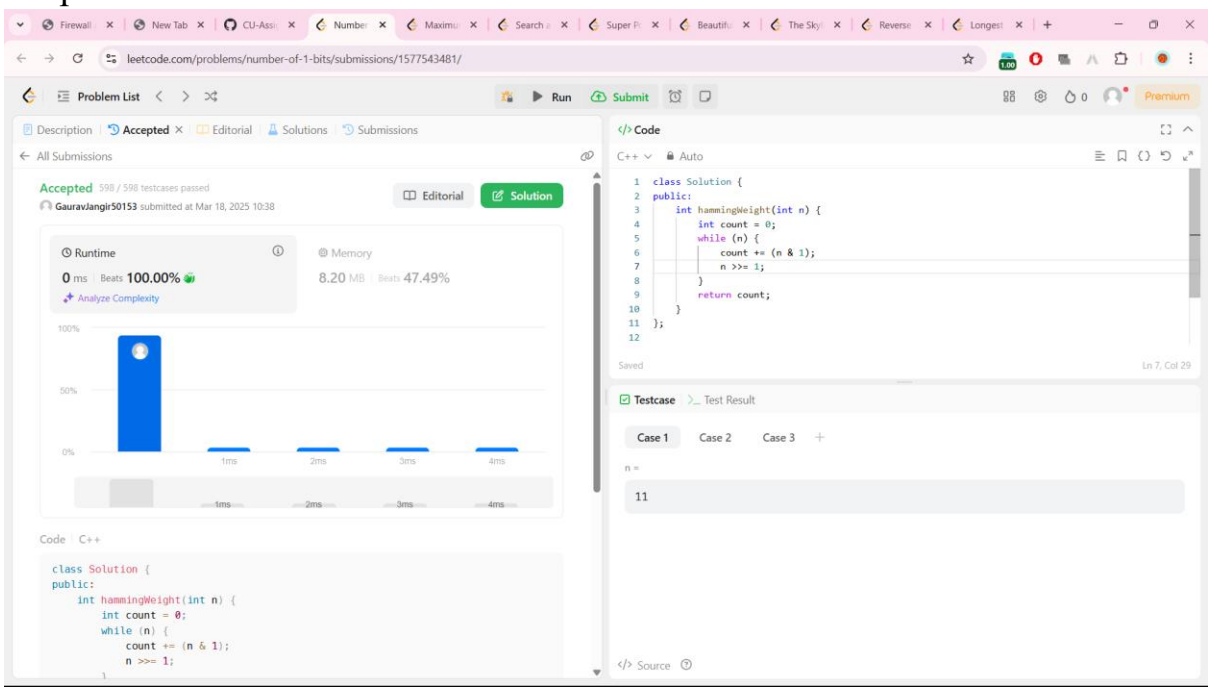
Output:



3. Number of 1 Bits:

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

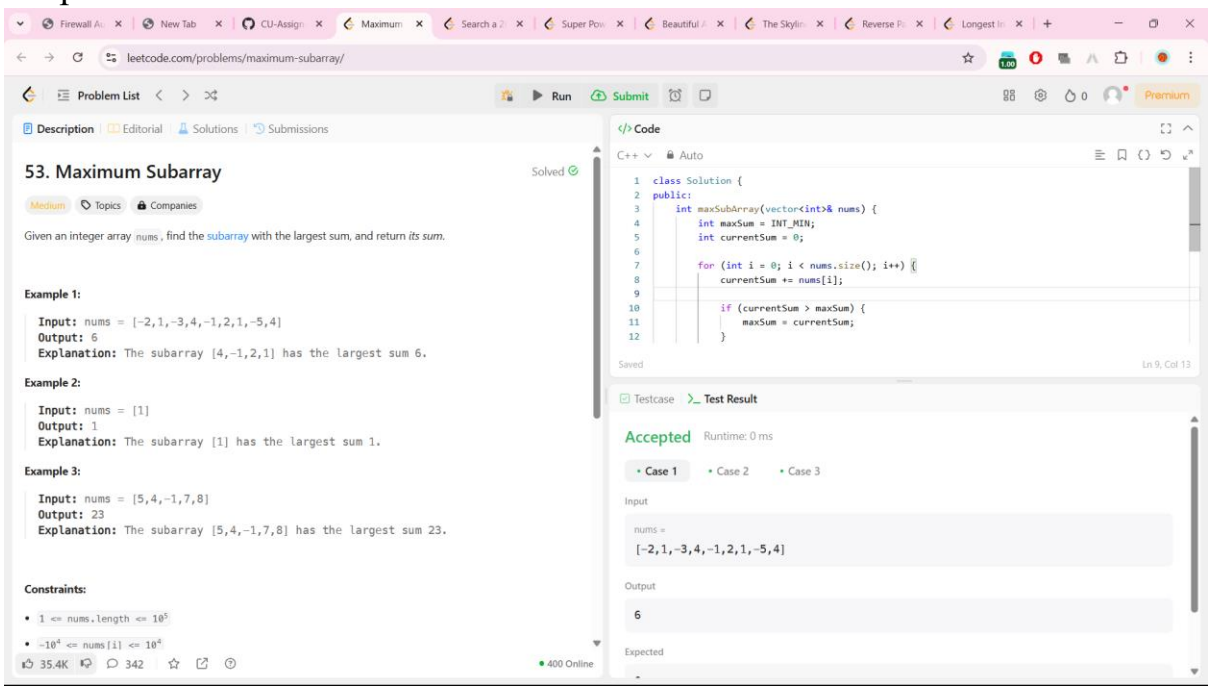
Output:



4. Maximum Subarray:

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

        for (int i = 0; i < nums.size(); i++) {
            currentSum += nums[i];

            if (currentSum > maxSum) {
                maxSum = currentSum;
            }

            if (currentSum < 0) {
                currentSum = 0;
            }
        }

        return maxSum;
    }
};
```

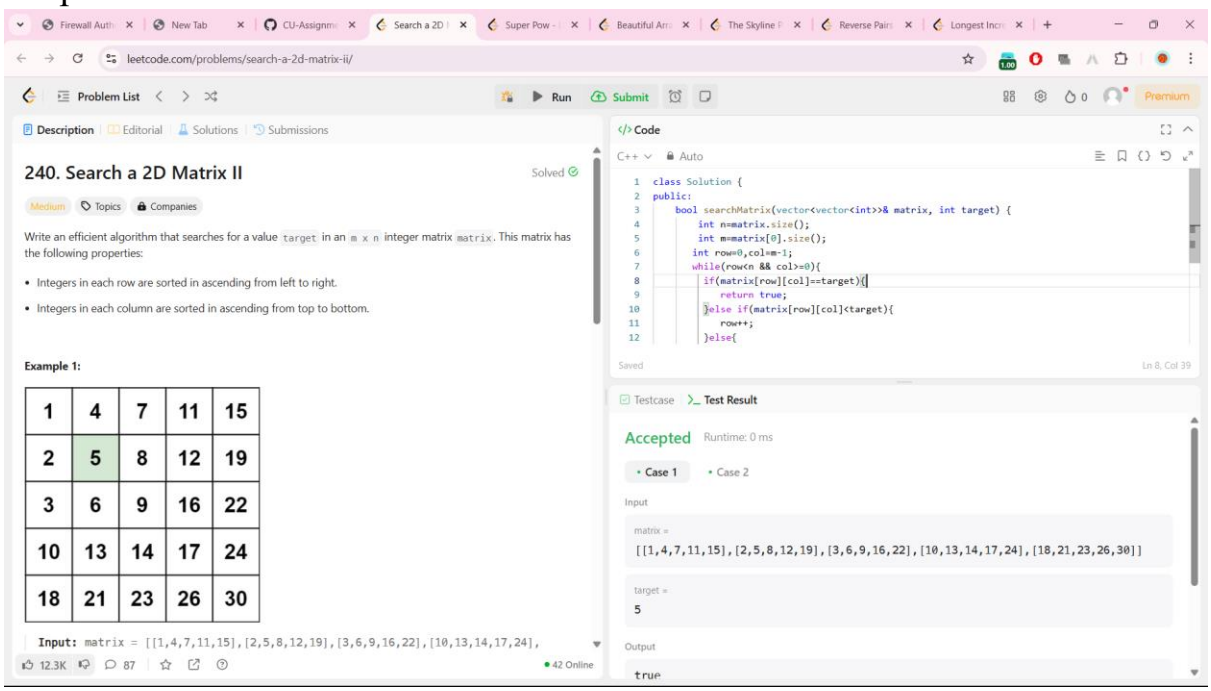
Output:



5. Search a 2D Matrix II:

```
class Solution {
public:
   bool searchMatrix(vector<vector<int>>& matrix, int target) {
       int n=matrix.size();
       int m=matrix[0].size();
      int row=0,col=m-1;
      while(row<n && col>=0){
        if(matrix[row][col]==target){
           return true;
        }else if(matrix[row][col]<target){
           row++;
        }else{
           col--;
        }
      }
      return false;
   }
};
```

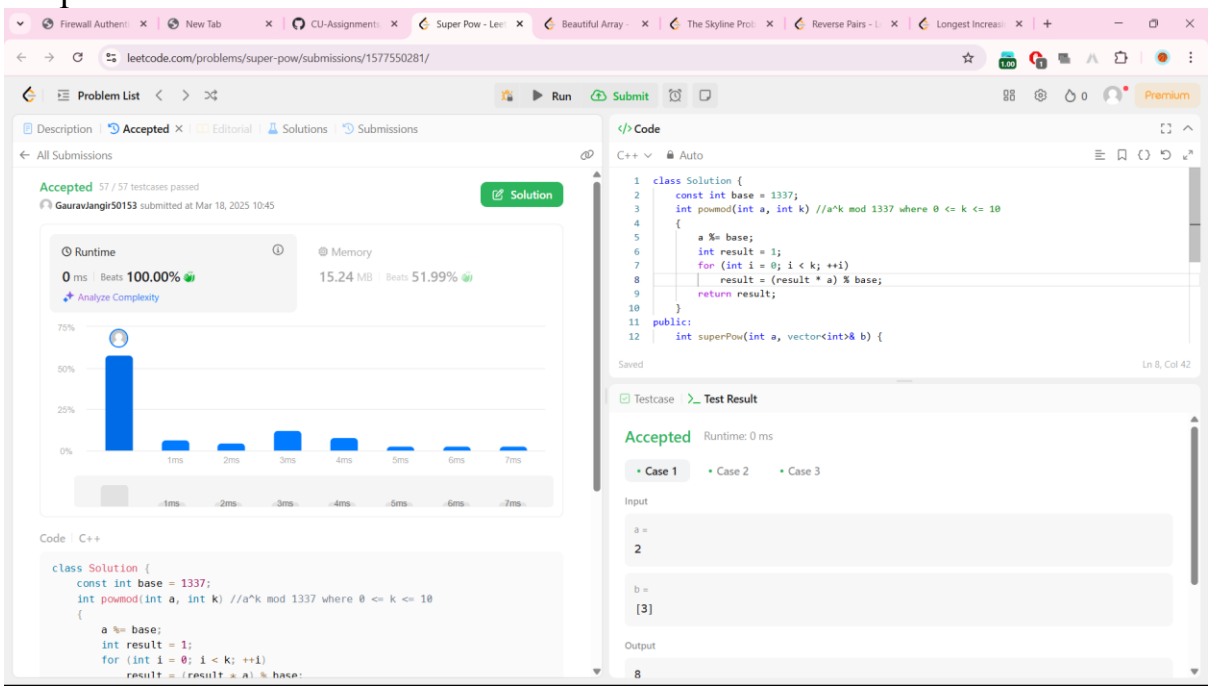
Output:



6. Super Pow:

```
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;
    }
};
```

Output:

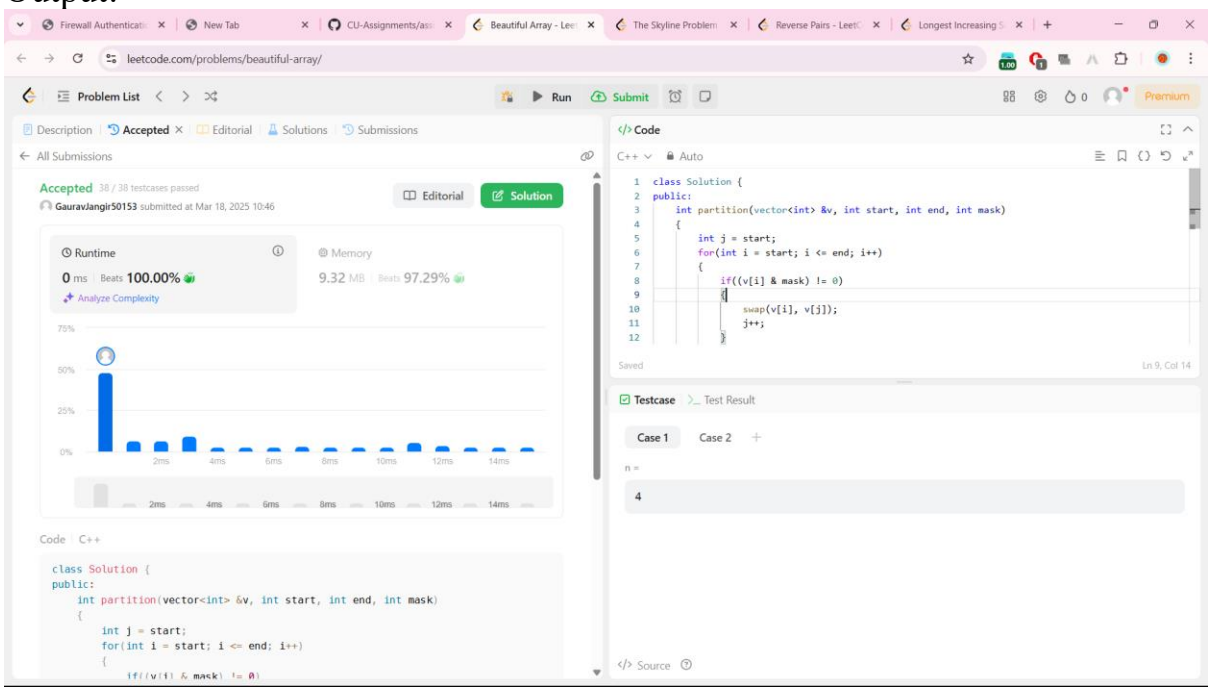


7. Beautiful Array:

```
class Solution {
public:
    int partition(vector<int> &v, int start, int end, int mask)
    {
        int j = start;
        for(int i = start; i <= end; i++)
        {
            if((v[i] & mask) != 0)
            {
                swap(v[i], v[j]);
                j++;
            }
        }
        return j;
    }

    void sort(vector<int> & v, int start, int end, int mask)
    {
        if(start >= end) return;
        int mid = partition(v, start, end, mask);
        sort(v, start, mid - 1, mask << 1);
        sort(v, mid, end, mask << 1);
```

```
    }

    vector<int> beautifulArray(int N) {
        vector<int> ans;
        for(int i = 0; i < N; i++) ans.push_back(i + 1);
        sort(ans, 0, N - 1, 1);
        return ans;
    }
};
```

Output:

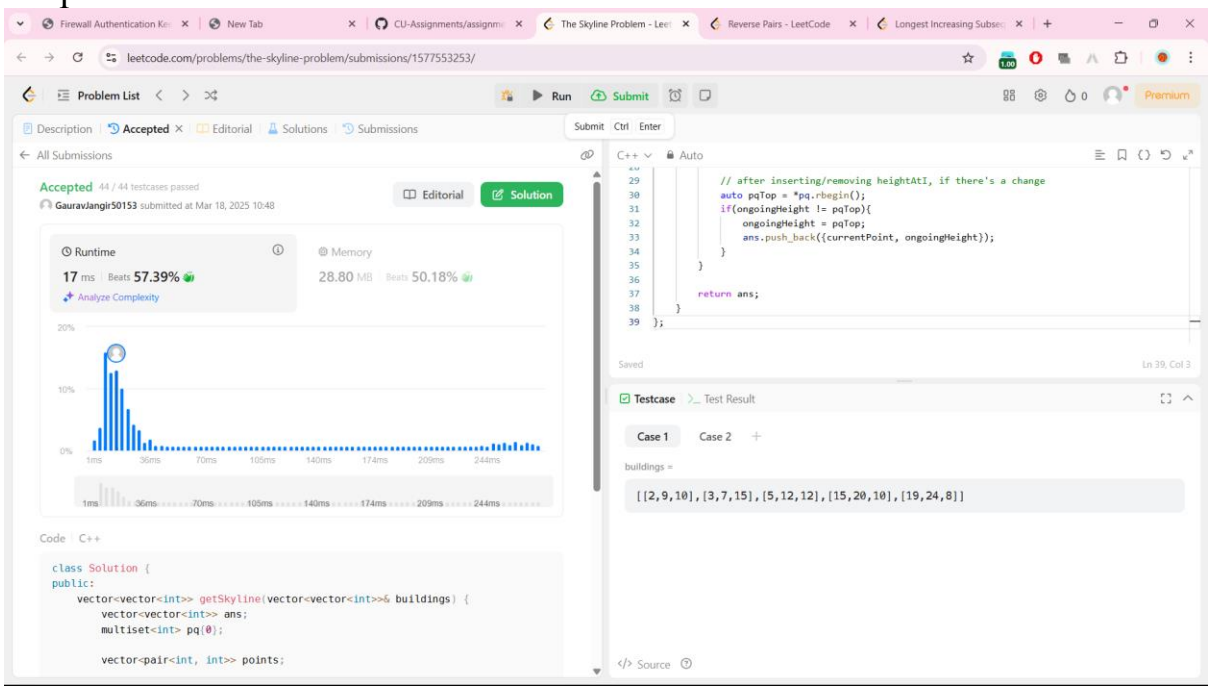


8. The Skyline Problem:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

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

```
        }

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

        int ongoingHeight = 0;

        // points.first = x coordinate, points.second = height
        for(int i = 0; i < points.size(); i++){
            int currentPoint = points[i].first;
            int heightAtCurrentPoint = points[i].second;

            if(heightAtCurrentPoint < 0){
                pq.insert(-heightAtCurrentPoint);
            } else {
                pq.erase(pq.find(heightAtCurrentPoint));
            }

            // after inserting/removing heightAtI, if there's a change
            auto pqTop = *pq.rbegin();
            if(ongoingHeight != pqTop){
                ongoingHeight = pqTop;
                ans.push_back({currentPoint, ongoingHeight});
            }
        }

        return ans;
    }
};
```

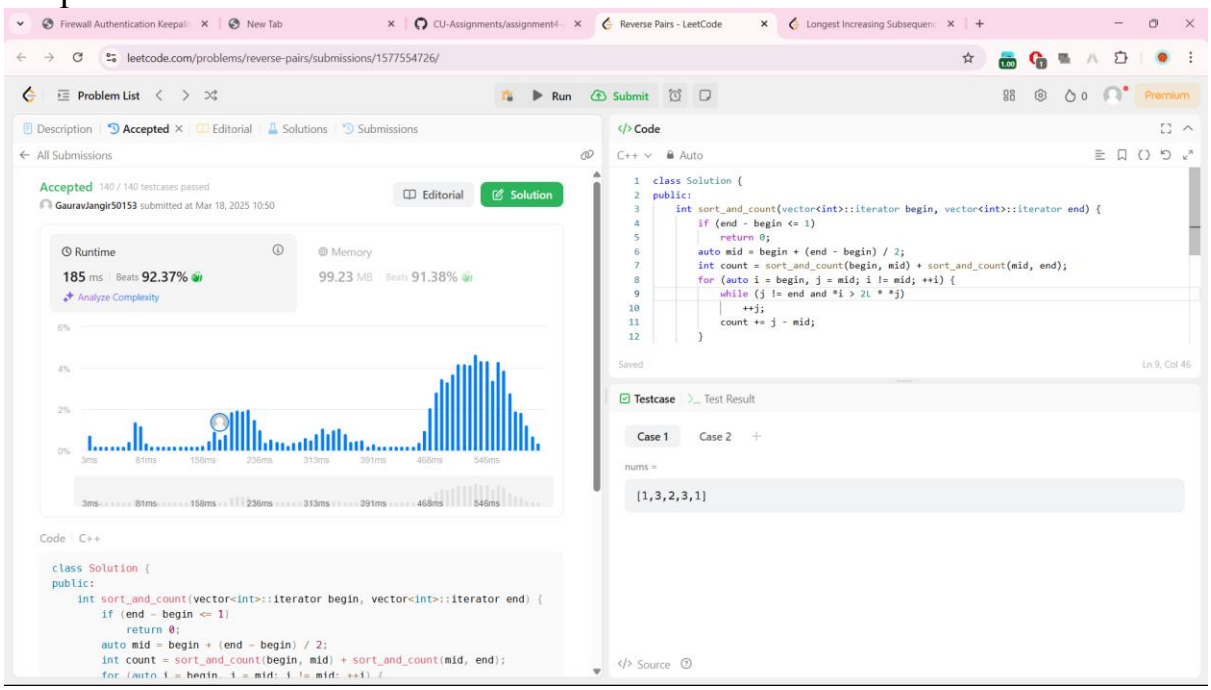
Output:



9. Reverse Pairs:

```
class Solution {
public:
    int sort_and_count(vector<int>::iterator begin, vector<int>::iterator end) {
        if (end - begin <= 1)
            return 0;
        auto mid = begin + (end - begin) / 2;
        int count = sort_and_count(begin, mid) + sort_and_count(mid, end);
        for (auto i = begin, j = mid; i != mid; ++i) {
            while (j != end and *i > 2L * *j)
                ++j;
            count += j - mid;
        }
        inplace_merge(begin, mid, end);
        return count;
    }

    int reversePairs(vector<int>& nums) {
        return sort_and_count(nums.begin(), nums.end());
    }
};
```

Output:



10. Longest Increasing Subsequence II:

```
class Solution {
public:
    vector<int> seg;
    //Segment tree to return maximum in a range
    void upd(int ind, int val, int x, int lx, int rx) {
        if(lx == rx) {
            seg[x] = val;
            return;
        }
        int mid = lx + (rx - lx) / 2;
        if(ind <= mid)
            upd(ind, val, 2 * x + 1, lx, mid);
        else
            upd(ind, val, 2 * x + 2, mid + 1, rx);
        seg[x] = max(seg[2 * x + 1], seg[2 * x + 2]);
    }
    int query(int l, int r, int x, int lx, int rx) {
        if(lx > r or rx < l) return 0;
        if(lx >= l and rx <= r) return seg[x];
```

```
        int mid = lx + (rx - lx) / 2;
        return max(query(l, r, 2 * x + 1, lx, mid), query(l, r, 2 * x + 2, mid + 1,
rx));
    }

    int lengthOfLIS(vector<int>& nums, int k) {
        int x = 1;
        while(x <= 200000) x *= 2;
        seg.resize(2 * x, 0);

        int res = 1;
        for(int i = 0; i < nums.size(); ++i) {
            int left = max(1, nums[i] - k), right = nums[i] - 1;
            int q = query(left, right, 0, 0, x - 1); // check for the element in the range
of [nums[i] - k, nums[i] - 1] with the maximum value
            res = max(res, q + 1);
            upd(nums[i], q + 1, 0, 0, x - 1); //update current value
        }
        return res;
    }
};
```

Output: