ASSIGNMENT - 4

Student Name: Tanisha Mahajan **UID: 22BCS11132**

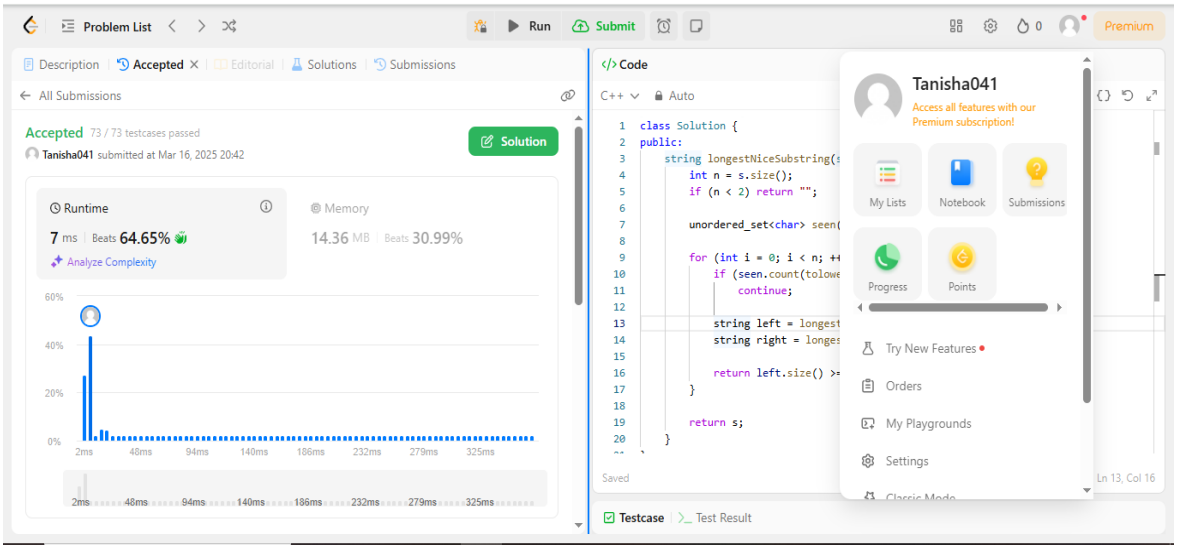
Branch: BE-CSE **Section/Group: 22BCS_TPP_607-B**

Semester: 6TH **Subject Name : AP-11**

Divide and Conquer :

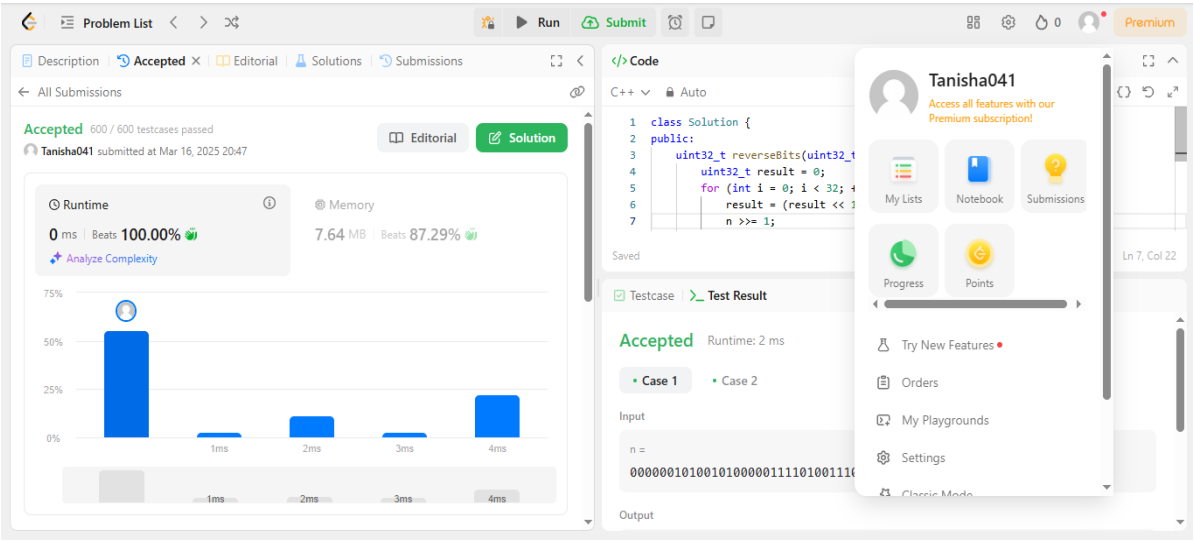
1. **Longest Nice Substring :**

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.size();
        if (n < 2) return "";
        unordered_set<char> seen(s.begin(), s.end());
        for (int i = 0; i < n; ++i) {
            if (seen.count(tolower(s[i])) && seen.count(toupper(s[i])))
                continue;
            string left = longestNiceSubstring(s.substr(0, i));
            string right = longestNiceSubstring(s.substr(i + 1));
            return left.size() >= right.size() ? left : right;
        }
        return s;
    }
};
```



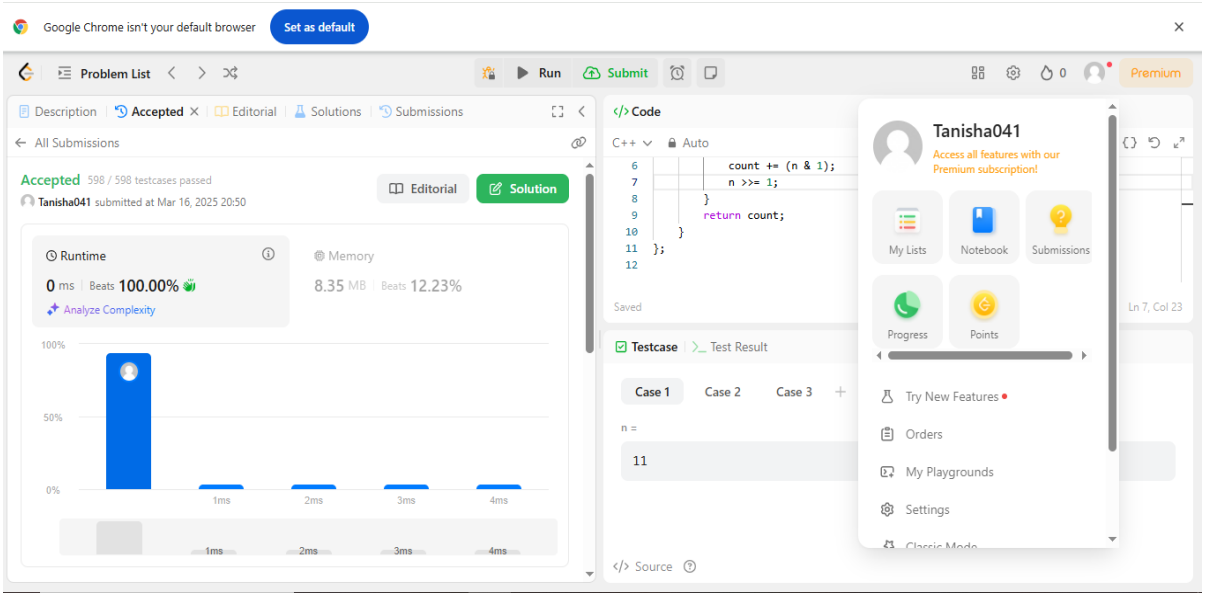
2. **Reverse Bits :**

```
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 Bit :**

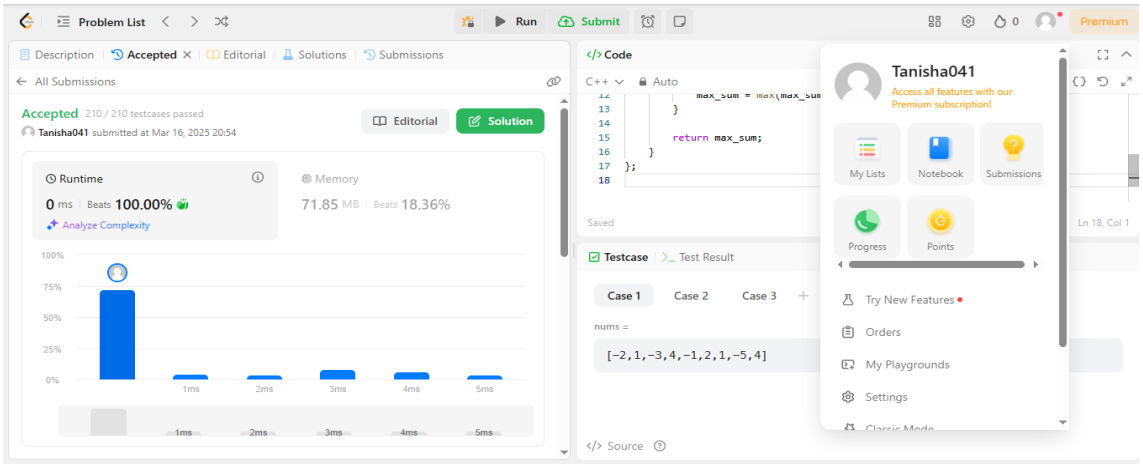
```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>= 1;
        }
        return count;
    }
};
```



4. **Maximum Subarray :**

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

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

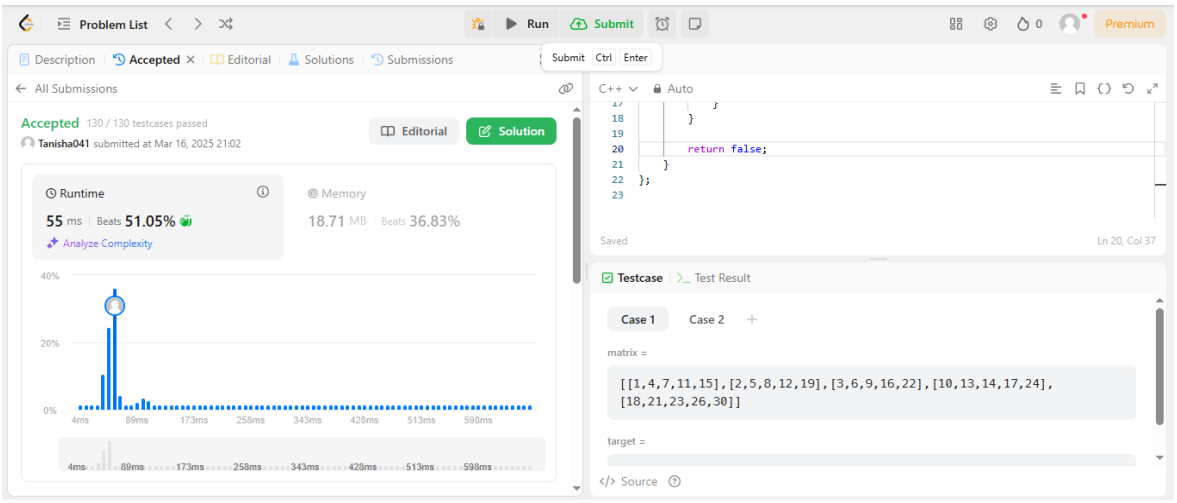


5. **Search a 2D Matrix II :**

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

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

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



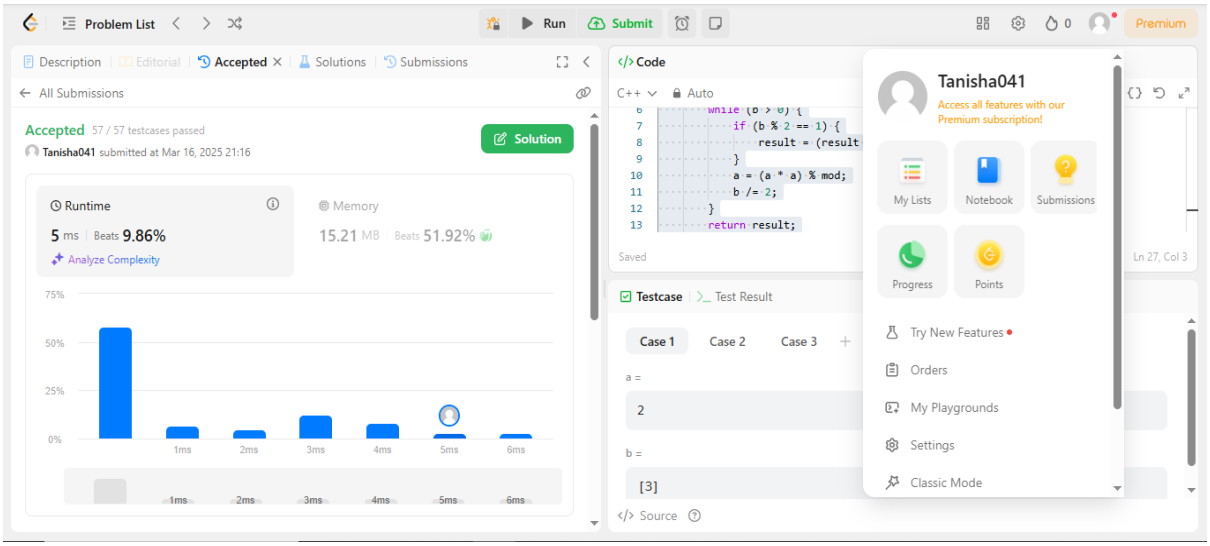
6. **Super Pow :**

```
class Solution {
public:
    int modPow(int a, int b, int mod) {
        int result = 1;
        a %= mod;
        while (b > 0) {
            if (b % 2 == 1) {
                result = (result * a) % mod;
            }
            a = (a * a) % mod;
            b /= 2;
        }
        return result;
    }

    int superPow(int a, vector<int>& b) {
        int mod = 1337;
        a %= mod;
        int result = 1;

        for (int digit : b) {
            result = modPow(result, 10, mod) * modPow(a, digit, mod) % mod;
        }

        return result;
    }
};
```

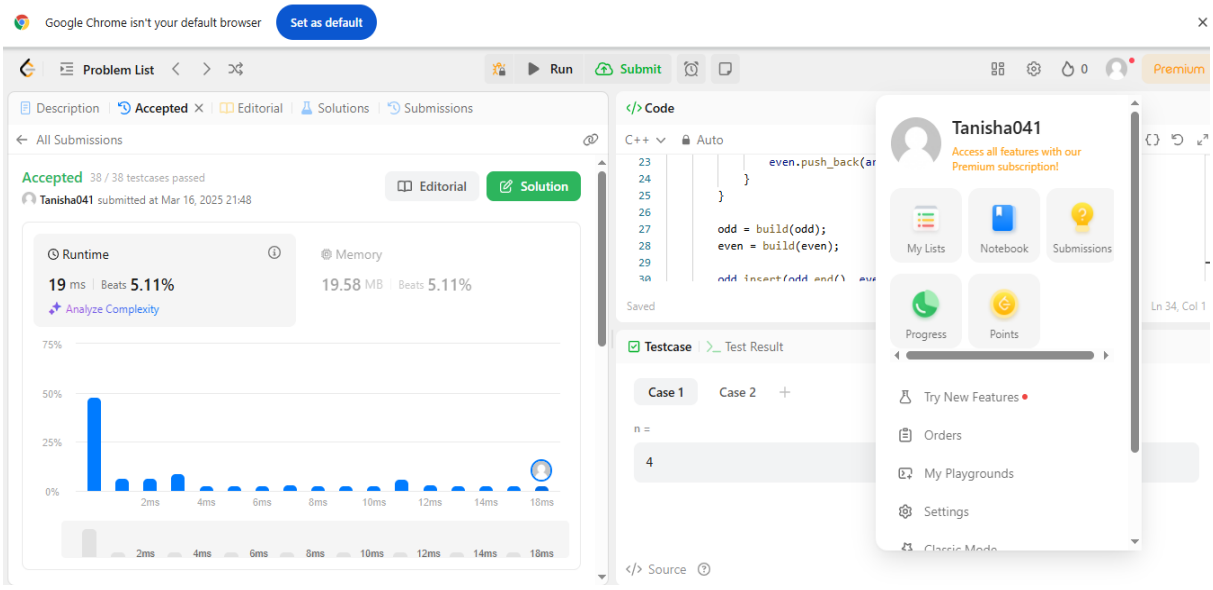


7. **Beautiful Array :**

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            arr[i] = i + 1;
        }
        return build(arr);
    }

private:
    vector<int> build(vector<int>& arr) {
        if (arr.size() <= 1) {
            return arr;
        }
        vector<int> odd, even;
        for (int i = 0; i < arr.size(); i++) {
            if (i % 2 == 0) {
                odd.push_back(arr[i]);
            } else {
                even.push_back(arr[i]);
            }
```

```
        }
        odd = build(odd);
        even = build(even);
        odd.insert(odd.end(), even.begin(), even.end());
        return odd;
    }
};
```



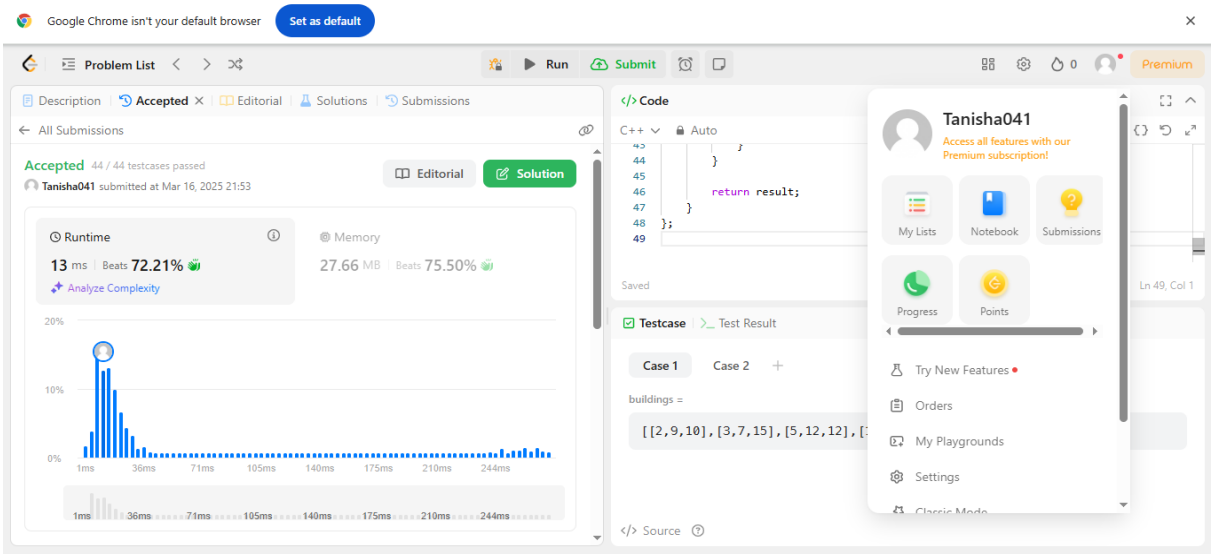
8. **The Skyline Problem :**

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        for (auto& building : buildings) {
            int left = building[0], right = building[1], height = building[2];
            events.push_back({left, -height});
            events.push_back({right, height});
        }

        sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
            if (a.first == b.first) {
                return a.second < b.second;
            }
```

```
            return a.first < b.first;
        });
        vector<vector<int>> result;
        multiset<int> heights = {0};
        int prevMaxHeight = 0;

        for (auto& event : events) {
            int x = event.first;
            int height = event.second;
            if (height < 0) {
                heights.insert(-height);
            } else {
                heights.erase(heights.find(height))
            }
            int currentMaxHeight = *heights.rbegin();
            if (currentMaxHeight != prevMaxHeight) {
                result.push_back({x, currentMaxHeight});
                prevMaxHeight = currentMaxHeight;
            }
        }
        return result;
    }
};
```

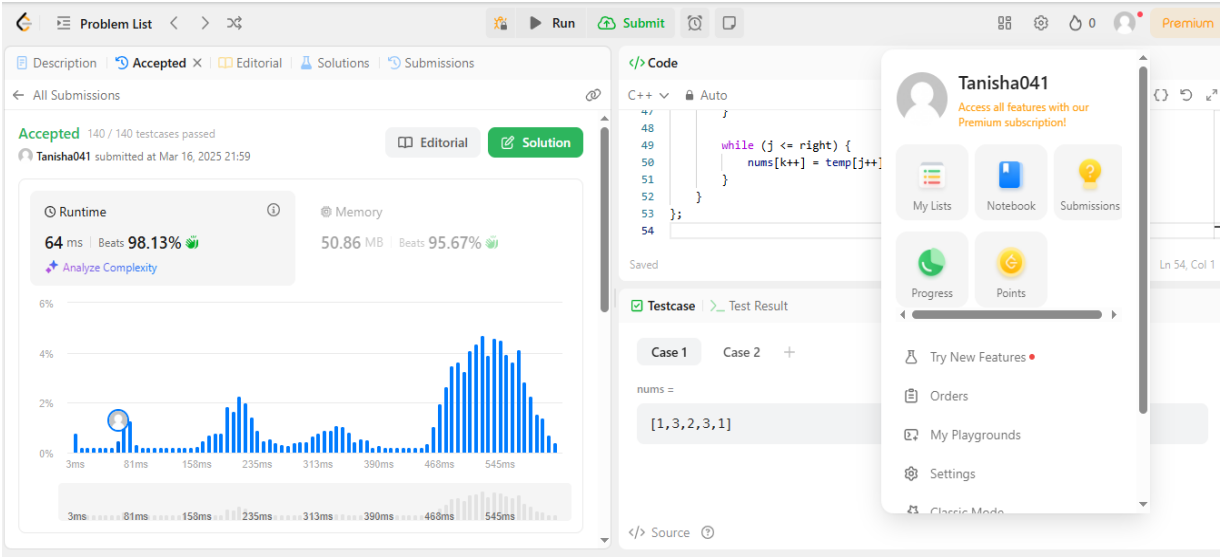


9. **Reverse Pairs :**

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

private:
    int mergeSort(vector<int>& nums, vector<int>& temp, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, temp, left, mid) + mergeSort(nums, temp, mid + 1, right);
        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && (long long)nums[i] > 2 * (long long)nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }
        merge(nums, temp, left, mid, right);
        return count;
    }

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



10. Longest Increasing Subsequence II :

```
class MaxSegmentTree {
 public:
  int n;
  vector<int> tree;
  MaxSegmentTree(int n_) : n(n_) {
    int size = (int)(ceil(log2(n)));
    size = (2 * pow(2, size)) - 1;
    tree = vector<int>(size);
  }

  int max_value() { return tree[0]; }
  int query(int l, int r) { return query_util(0, l, r, 0, n - 1); }
  int query_util(int i, int qL, int qR, int l, int r) {
    if (l >= qL && r <= qR) return tree[i];
    if (l > qR || r < qL) return INT_MIN;
    int m = (l + r) / 2;
    return max(query_util(2 * i + 1, qL, qR, l, m), query_util(2 * i + 2, qL, qR, m + 1, r));
  }

  void update(int i, int val) { update_util(0, 0, n - 1, i, val); }
  void update_util(int i, int l, int r, int pos, int val) {
    if (pos < l || pos > r) return;
    if (l == r) {
      tree[i] = max(val, tree[i]);
```

```
      return;
    }
    int m = (l + r) / 2;
    update_util(2 * i + 1, l, m, pos, val);
    update_util(2 * i + 2, m + 1, r, pos, val);
    tree[i] = max(tree[2 * i + 1], tree[2 * i + 2]);
  }
};

class Solution {
 public:
  int lengthOfLIS(vector<int>& nums, int k) {
    MaxSegmentTree tree(1e5 + 1);
    for (int i : nums) {
      int lower = max(0, i - k);
      int cur = 1 + tree.query(lower, i - 1);
      tree.update(i, cur);
    }
    return tree.max_value();
  }
};
```