WORKSHEET

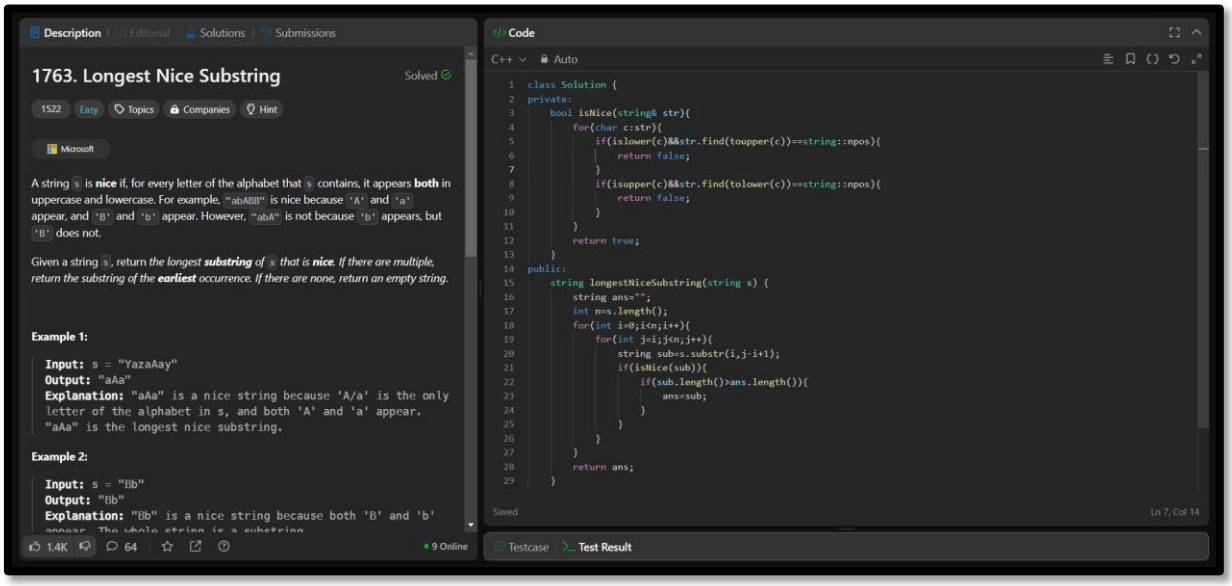
Student Name: Ishu **UID: 22BCS15695**

Branch: CSE **Section/Group: 609-'B'**

Semester: 6th **Date of Submission:26/07/24**

Subject Name: AP LAB

1. Longest Nice Substring:



class Solution {

private:

bool isNice(string str){

for(char c:str){

if(islower(c)88str.find(toupper(c))==string::npos) {

return false;}

if(isupper(c)8&str.find(tolower(c))==string::npos)(

return false;

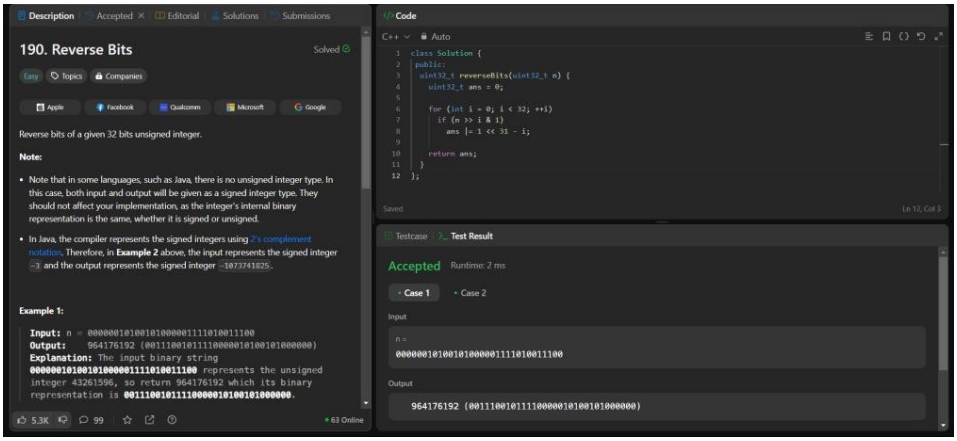
}

return true;

public:

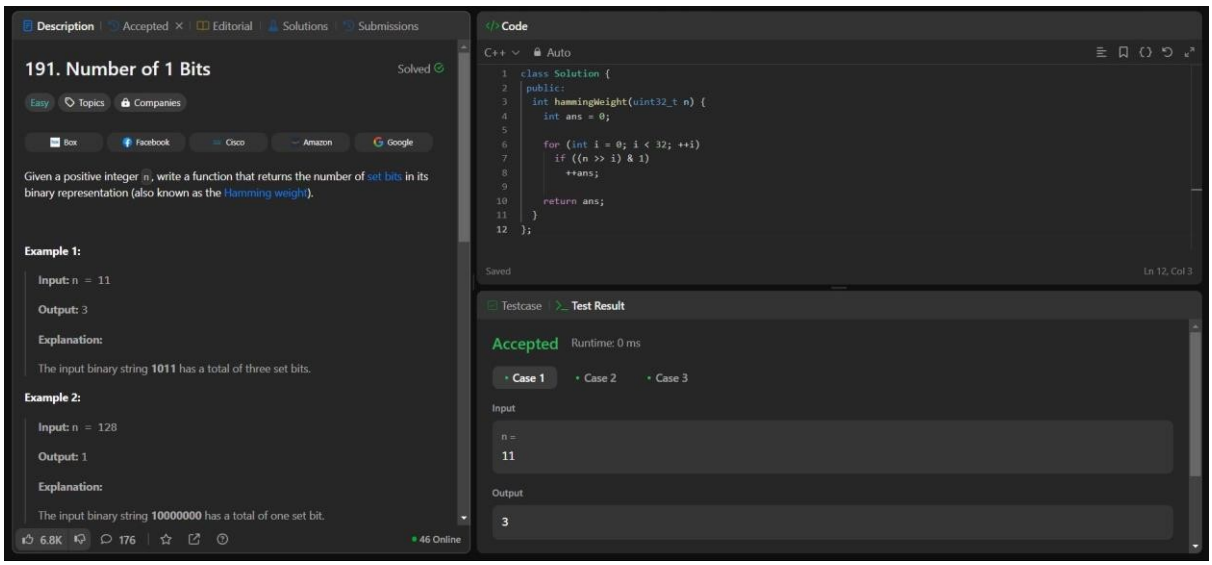
```
string longestNiceSubstring(string s)
string ans="";
int nos.length();
for(int i=0;i<n;i++){
for(int j=i;j<n;j++){
string sub-s.substr(i,j-i+1);
if(islice(sub)){
if(sub.length()>ans.length())
ans=sub;}
    }
  }
}
return ans;
}
```

2. Reverse Bits



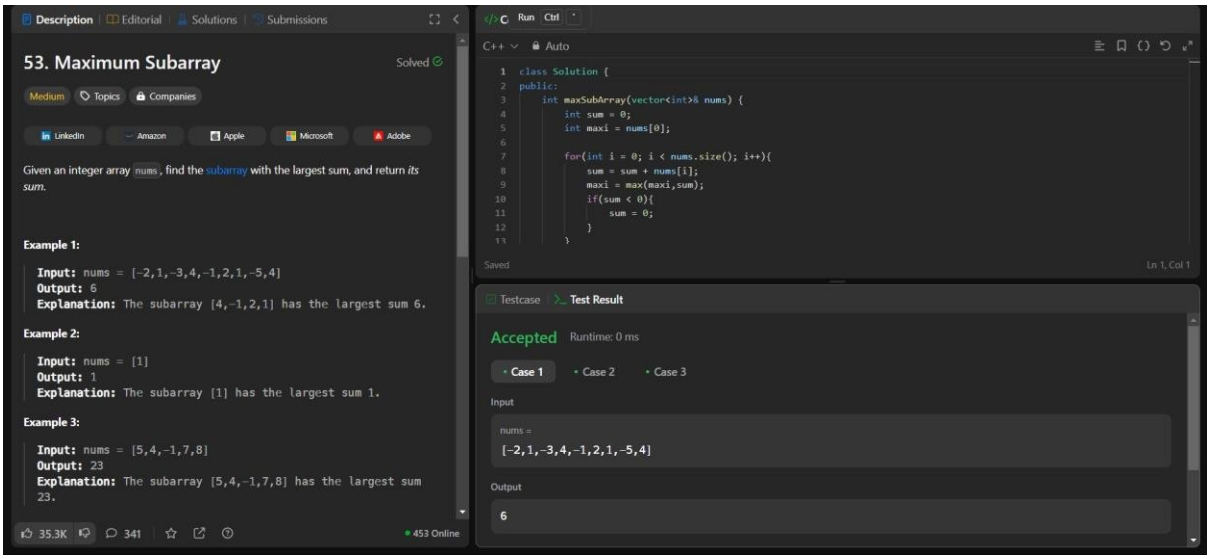
```
class Solution {
public:
uint32 t reverseBits(uint32_t n) {
uint32 t ans = 0;
for (int i=0; i < 32; ++i){
if (n >> i & 1){
ans = 1 << 31-;
return ans;}
}
```

3. Number of 1 Bits:



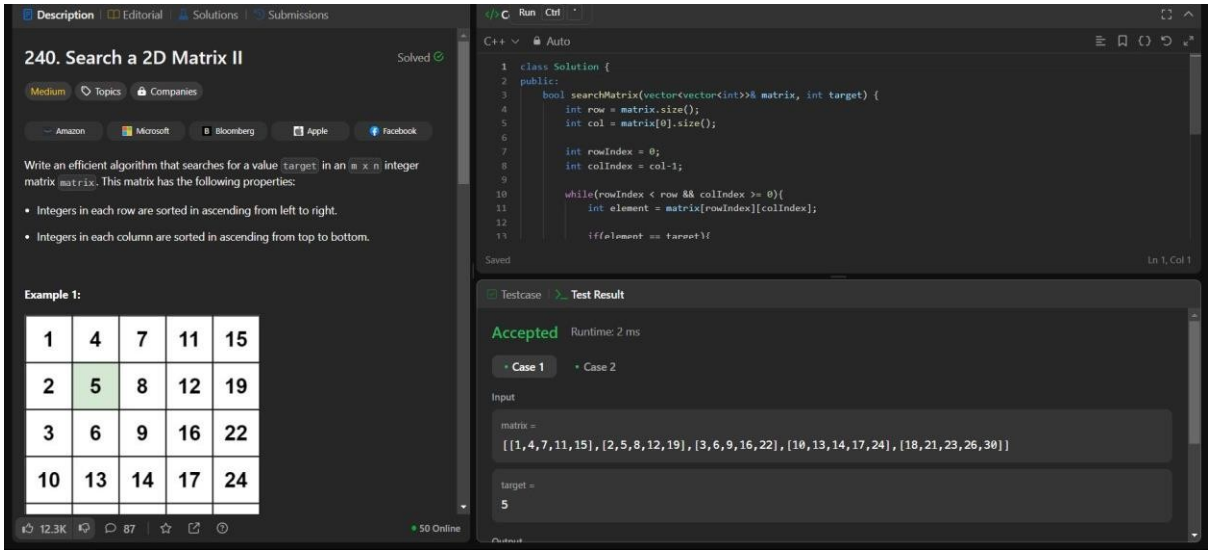

```
class Solution {
public:
int hammingWeight(uint32_t n) {
int ans = 0;
for (int i = 0; i < 32; ++i){
if ((n >> i) & 1)
++ans;
return ans;}
};
```

4. Maximum Subarray:



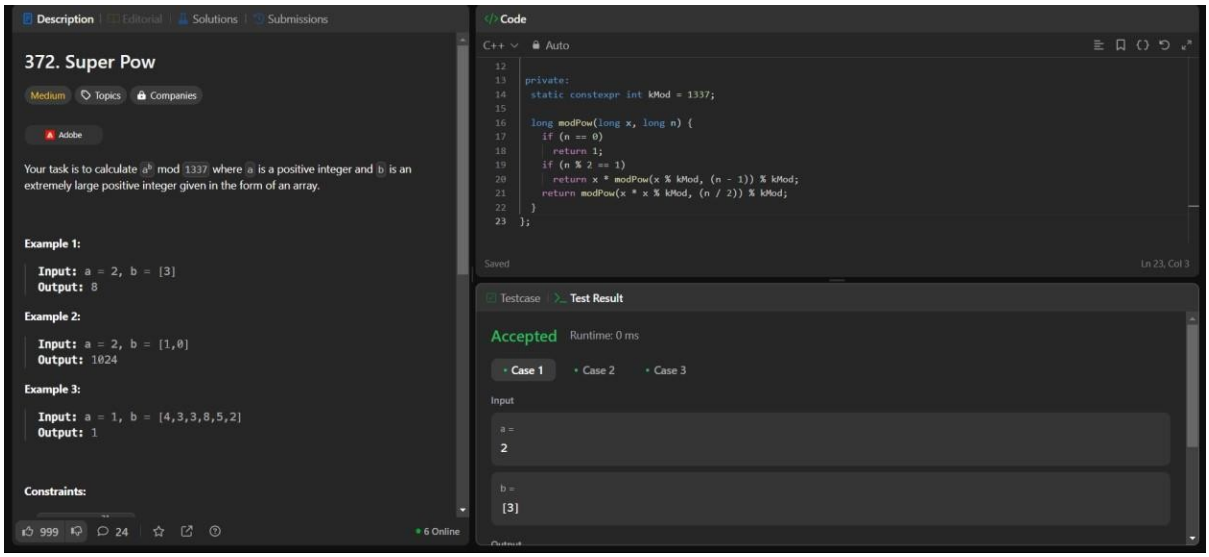

```
class Solution {
public:
int maxSubArray(vector<int>& nums) {
 int sum = 0;
int maxi = nums[0];
for(int i = 0; i < nums.size(); i++){
sum = sum + nums[i];
maxi = max(maxi, sum);
if(sum < 0){
sum = 0;}
}
```

5. Search a 2D Matrix II:



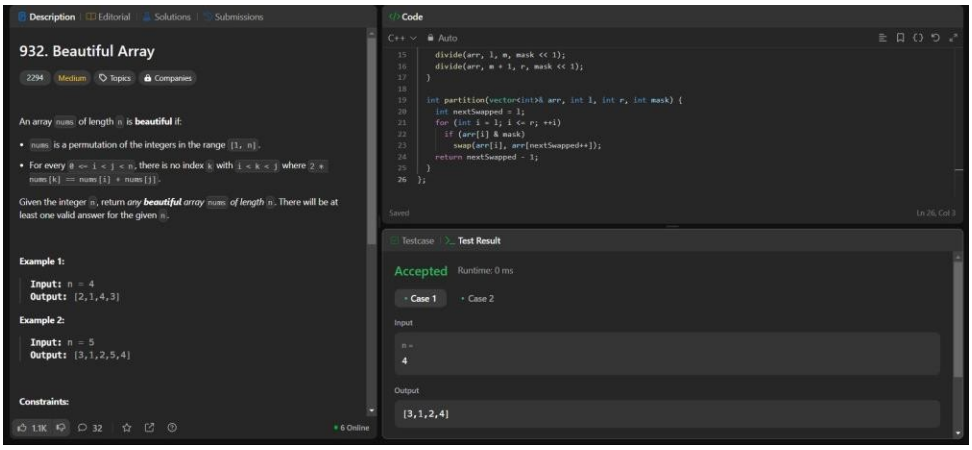

```
class Solution {
public: bool searchMatrix(vector<vector<int>>& matrix, int target) {
for (int i = 0; i < matrix.size(); i++) {
for (int j = 0; j < matrix[i].size(); j++) {
 if (matrix[i][j] == target) {
return true; }
 } }
return false; } };
```

6. Super Pow:



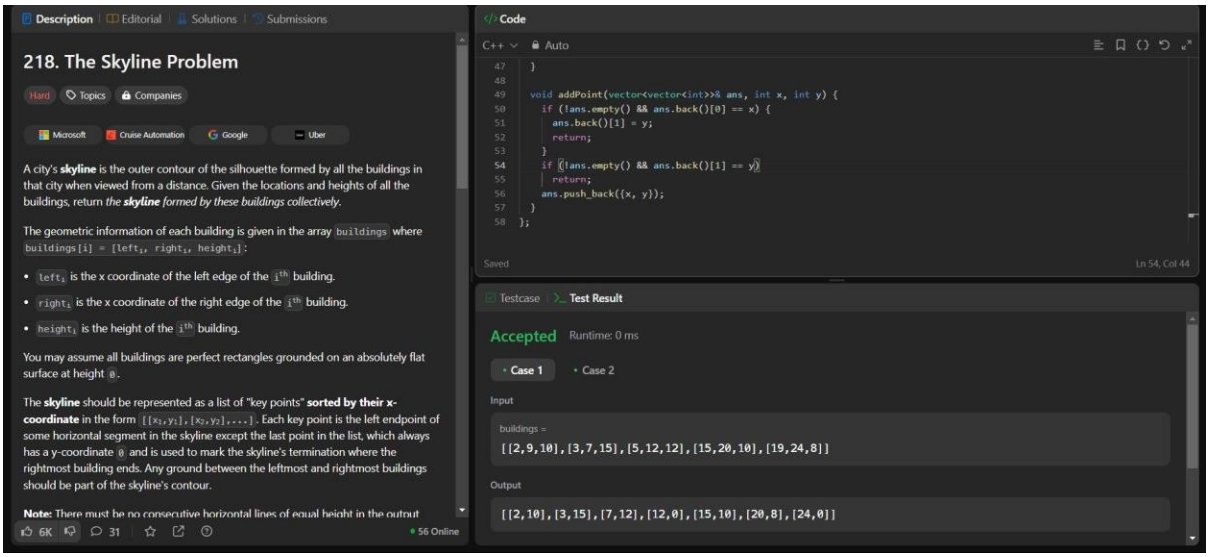

```
class Solution {
 private: int solve(int base, int power, int mod) {
int ans = 1; while (power > 0) {
 if (power & 1) { ans = (ans * base) % mod; }
base = (base * base) % mod; power >>= 1; }
return ans; }
public: int superPow(int a, vector<int>& b) {
a%=1337; int n = b.size();
int m = 1140; int expi = 0; for(int i : b){ expi = (expi*10+i)%m; }
 if (expi == 0) { expi = m;
}
return solve(a,expi,1337); } };
```

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; } };
```

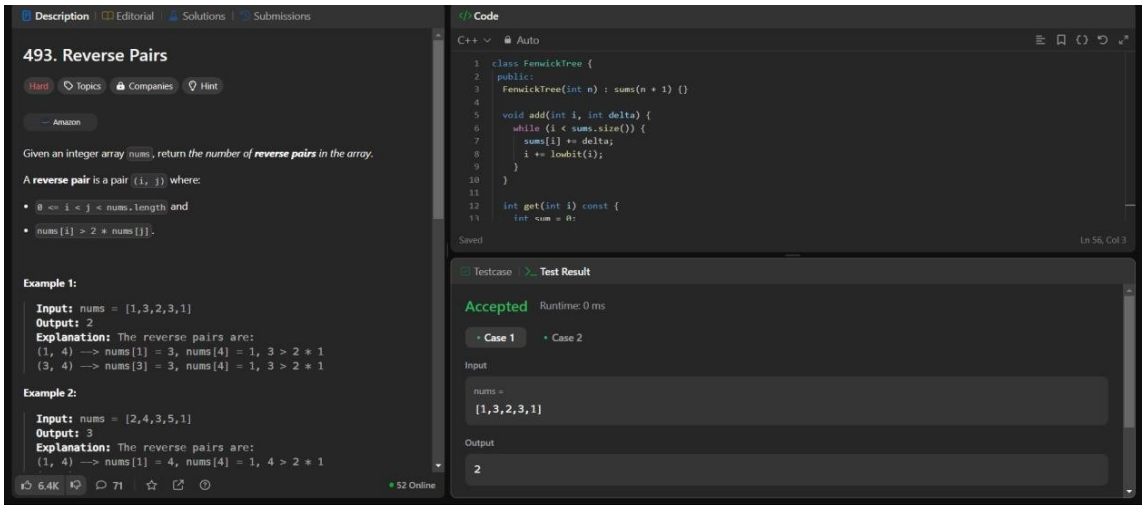
8. The Skyline Problem:




```
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; } };
```

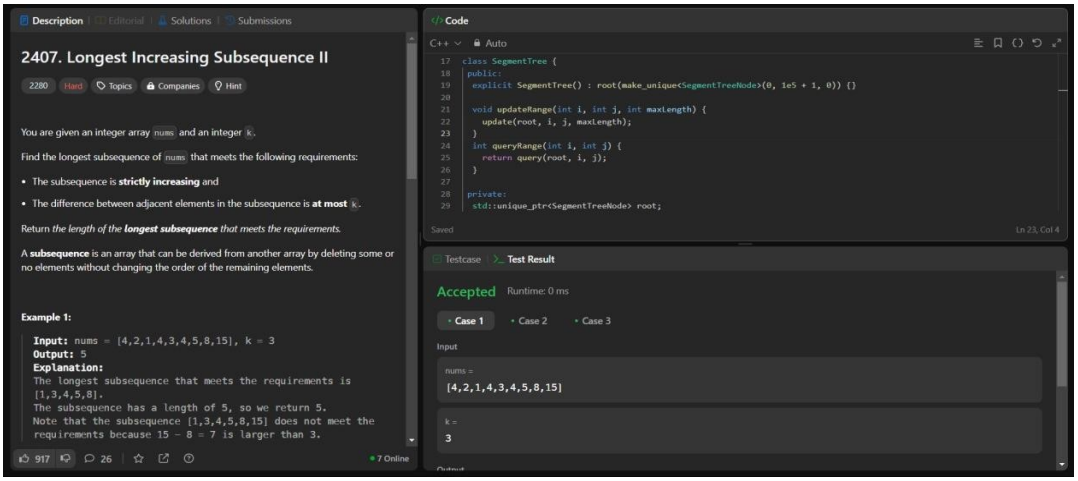
9. Reverse Pairs:




```
class Solution {
private:     void merge(vector<int>& nums, int low, int mid, int high, int&
reversePairsCount){
int j = mid+1;
for(int i=low; i<=mid; i++){
while(j<=high && nums[i] > 2*(long long)nums[j]){
```

```
j++; }
 reversePairsCount += j-(mid+1);}
int size = high-low+1;
vector<int> temp(size, 0);
int left = low, right = mid+1, k=0;
while(left<=mid && right<=high){
 if(nums[left] < nums[right]){
 temp[k++] = nums[left++];}
 else{temp[k++] = nums[right++];  }        }
while(left<=mid){            temp[k++] = nums[left++];         }
while(right<=high){            temp[k++] = nums[right++];         }
 int m=0;
for(int i=low; i<=high; i++){
 nums[i] = temp[m++];        }    }
 void mergeSort(vector<int>& nums, int low, int high, int& reversePairsCount){
if(low >= high){
 return;        }
 int mid = (low + high) >> 1;
mergeSort(nums, low, mid, reversePairsCount);
mergeSort(nums, mid+1, high, reversePairsCount);
merge(nums, low, mid, high, reversePairsCount);    }
public:    int reversePairs(vector<int>& nums) {
int reversePairsCount = 0;
mergeSort(nums, 0, nums.size()-1, reversePairsCount);
 return reversePairsCount;    }};
```

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]);}
};
classSolution{
```

```
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(); } };
```