Experiment 4

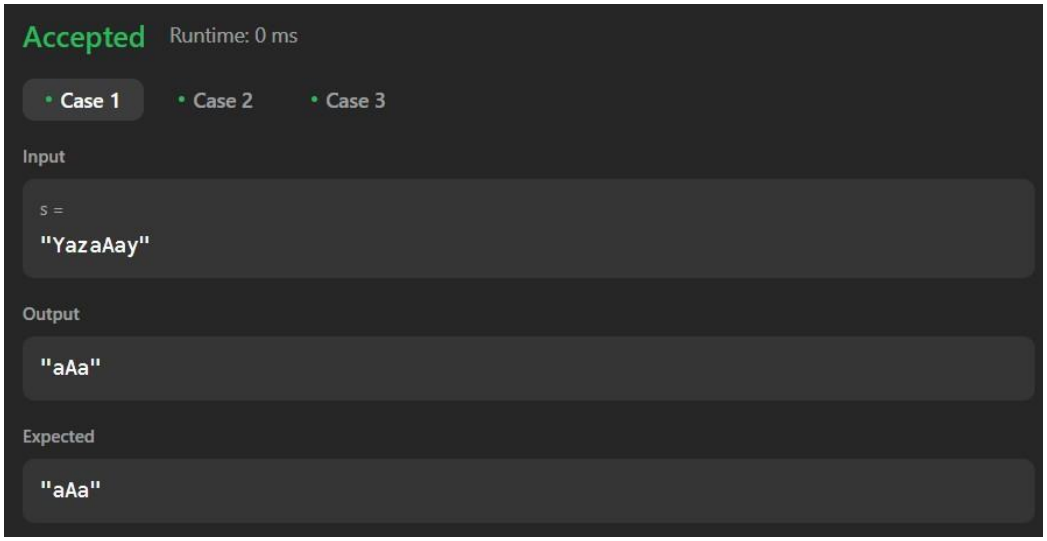
Name: Aryan Anand **UID: 22BET10056**
Branch: BE-IT **Section/Group: 22BET_702- B**
Semester: 6 **Date of Performance: 14-02-25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22ITP-351**

Problem 1. Longest Nice Substring - A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Code:

```
class Solution
{ public:
string longestNiceSubstring(string s) {        if (s.size() < 2) return "";
unordered_set<char> st(begin(s), end(s));        for (int i = 0; i < s.size(); i++) {            if
(st.find((char) toupper(s[i])) == end(st) || st.find((char) tolower(s[i])) == end(st)) {
string s1 = longestNiceSubstring(s.substr(0, i));            string s2 =
longestNiceSubstring(s.substr(i + 1));            return s1.size() >= s2.size() ? s1 : s2;
        }
}
    return s;
  }
};
```

Output:



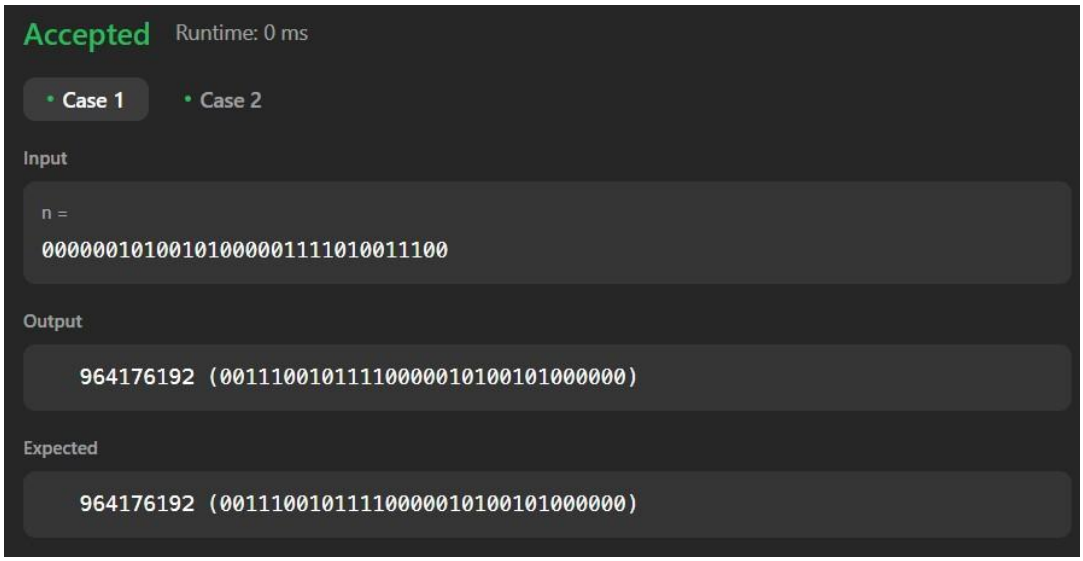
Problem 2.

Reverse Bits - Reverse bits of a given 32 bits unsigned integer

Code:

```
class Solution { public:     uint32_t reverseBits(uint32_t n) {
uint32_t result = 0;         for (int i = 0; i < 32; i++) {             int
bit = n & 1;       // Extract the least significant bit             result
= (result << 1) | bit; // Append the bit to the result             n = n
>> 1;          // Right-shift n to process the next bit
        }         return
result;
    }
};
```

Output:



Number of 1 bits. - Given a positive integer n, write a function that returns the number of set

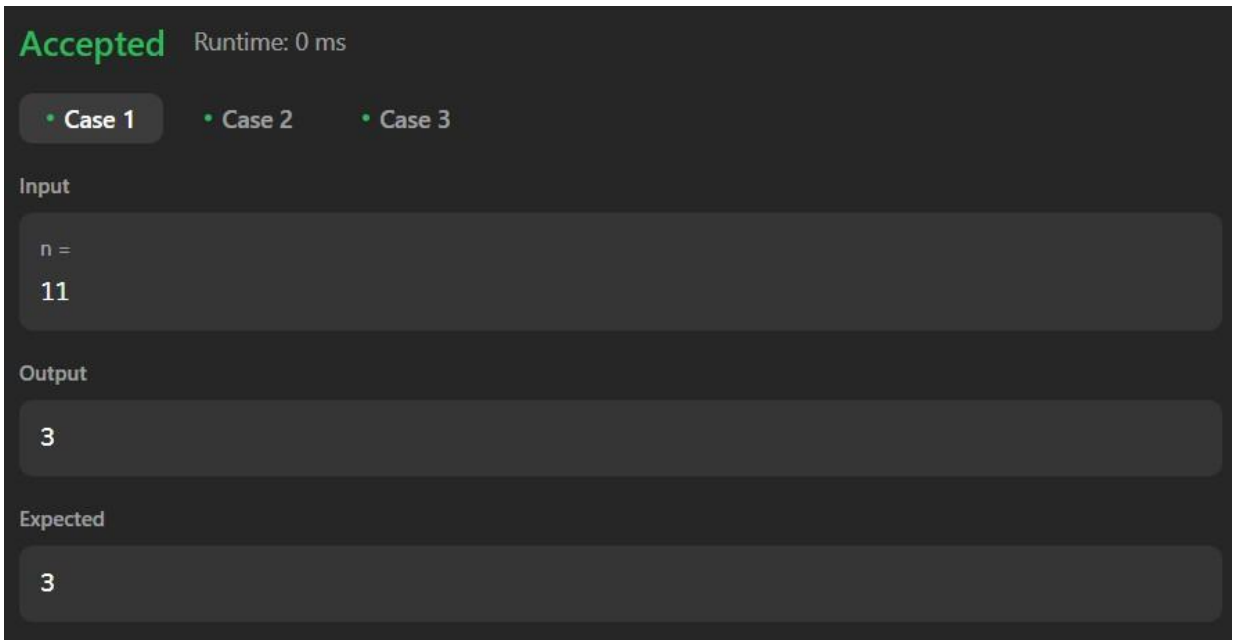
Problem 3.

bits in its binary representation (also known as the Hamming weight).

Code:

```
class Solution { public:     int
hammingWeight(int n) {
int count = 0;         while(n
>0){            if(n%2 ==1){
count++;                n = n/2;
          }
else{                n
=n/2;
          }
       }        return
count;
   } };
```

Output:



Max Subarray - Given an integer array nums, find the Subarray with the largest sum, and

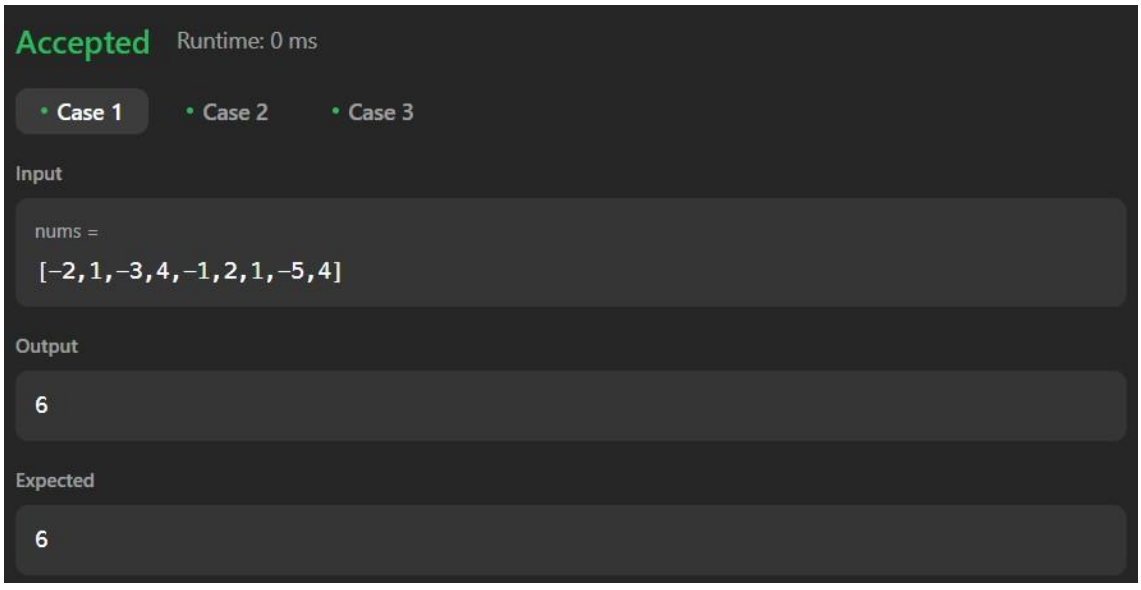
Problem 4.

return its sum.

Code:

```
class Solution { public:
   int maxSubArray(vector<int>& nums) {
int n = size(nums), ans = INT_MIN;
for(int i = 0; i < n; i++)            for(int j = i,
curSum = 0; j < n ; j++)                curSum
+= nums[j],              ans = max(ans,
curSum);              return ans;
   } };
```

Output:



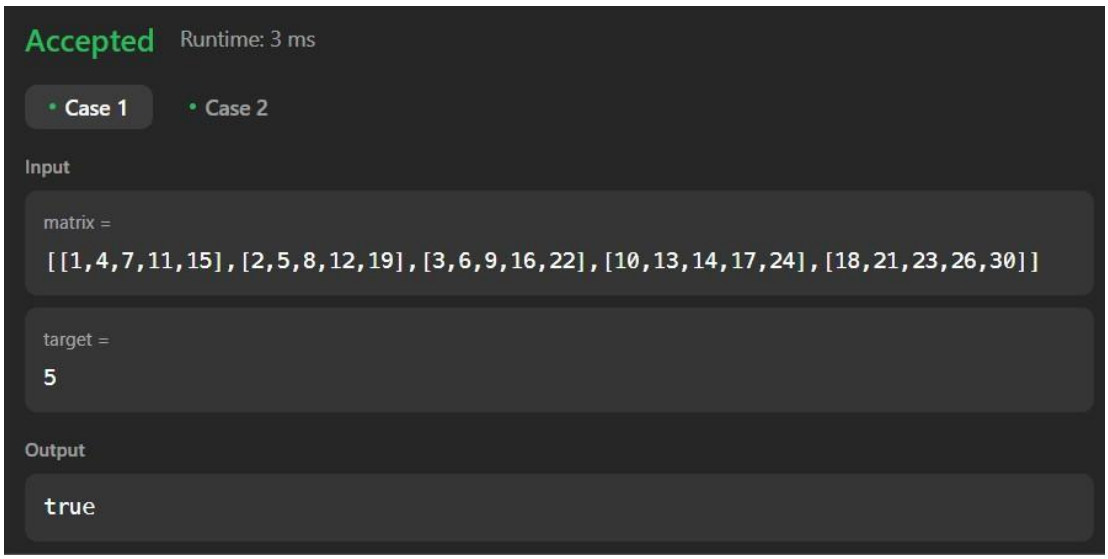
Problem 5.

Search 2d matrix 2 - Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Code:

```
class Solution { private:     bool
search(vector<int>& arr, int target) {         int
low = 0, high = arr.size() - 1;         while (low
<= high) {             int mid = low + (high - low)
/ 2;             if (arr[mid] == target) return true;
else if (arr[mid] < target) low = mid + 1;
else high = mid - 1;
        }
return false;
    }
public:     bool searchMatrix(vector<vector<int>>& matrix, int
target) {         int n = matrix.size(), m = matrix[0].size();
for (int i = 0; i < n; i++) {             if (search(matrix[i], target)) {
return true;
            }         }
return false;
    }
};
```

Output:



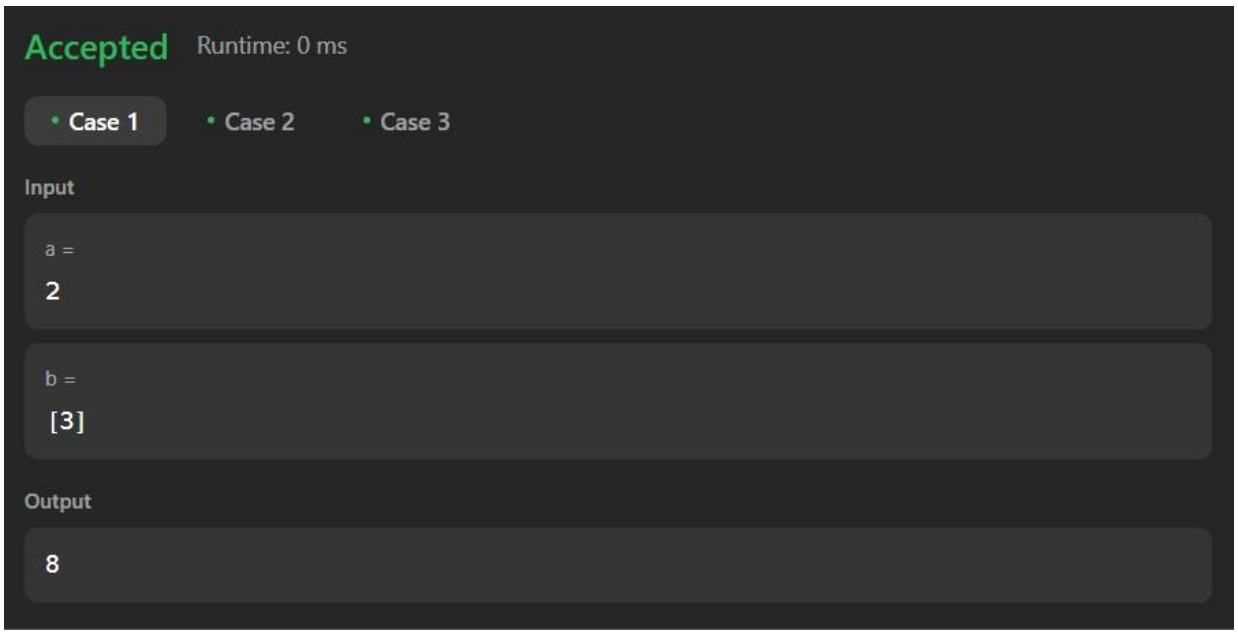
Problem 6.

Super Pow-Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Code:

```
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:



Beautiful Array- An array nums of length n is beautiful if:

Problem 7.

- nums is a permutation of the integers in the range [1, n].
- For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

Code:

```
class Solution { public:     vector<int> beautifulArray(int N) {
vector<int> res = {1};         while (res.size() < N) {
vector<int> tmp;             for (int i : res) if (i * 2 - 1 <= N)
tmp.push_back(i * 2 - 1);              for (int i : res) if (i * 2 <= N)
tmp.push_back(i * 2);             res = tmp;
      }
      return res;
   } };
```

Output:

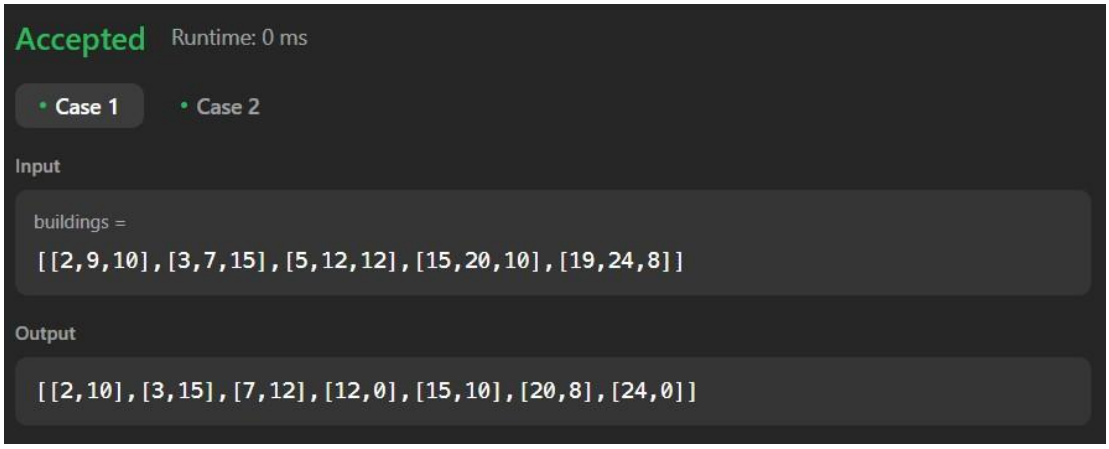
The Skyline Problem-A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return the skyline formed by these buildings collectively.

Problem 8.

Code:

```
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;          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));}            auto
pqTop = *pq.rbegin();           if(ongoingHeight !=
pqTop){               ongoingHeight = pqTop;
ans.push_back({currentPoint, ongoingHeight});
          }
}
      return ans;
   }
};
```

Output:



Reverse Pairs- Given an integer array nums, return the number of reverse pairs in the array.

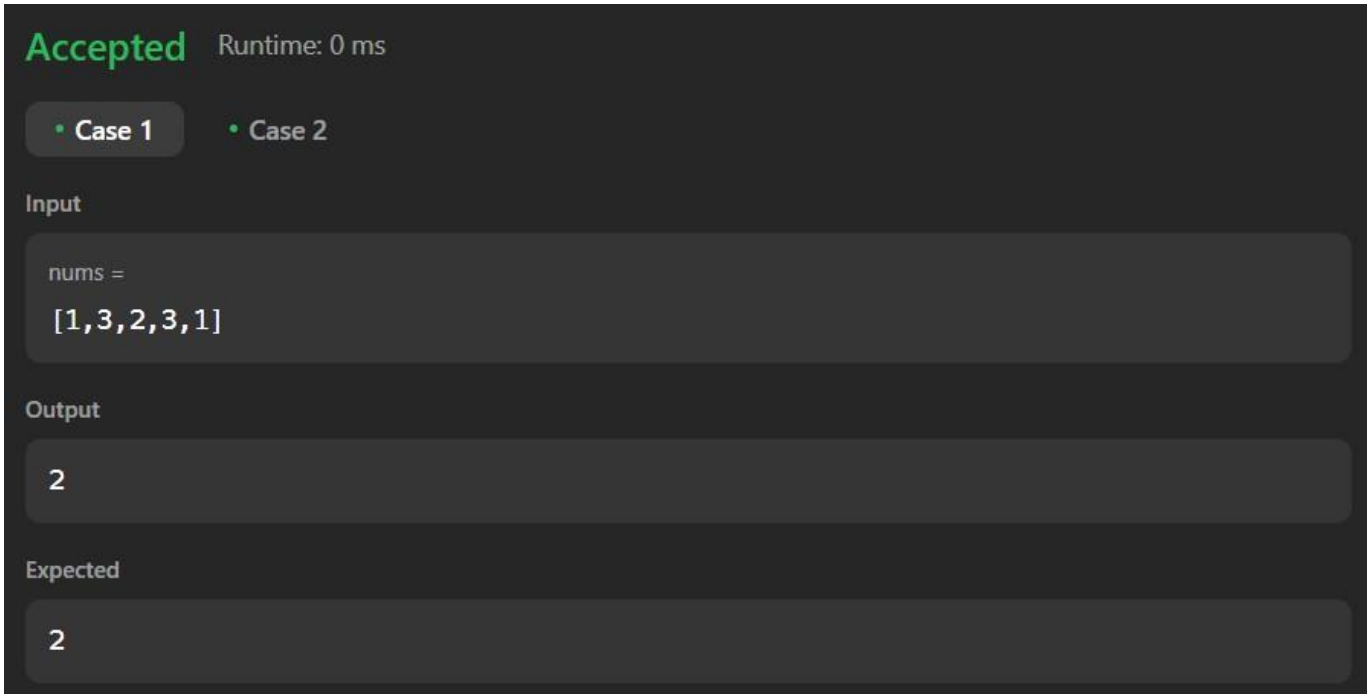
Code:

```
class Solution { public:    int
reversePairs(vector<int>& nums) {        int n
```

Problem 9.

```
= nums.size();        long long
reversePairsCount = 0;        for(int i=0; i<n-
1; i++){            for(int j=i+1; j<n; j++){
if(nums[i] > 2*(long long)nums[j]){
reversePairsCount++;
            }
          }
       }
       return reversePairsCount;
   } };
```

Output:



Longest increasing subsequence 2- You are given an integer array nums and an integer k.Find the longest subsequence of nums that meets the following requirements:

Code:

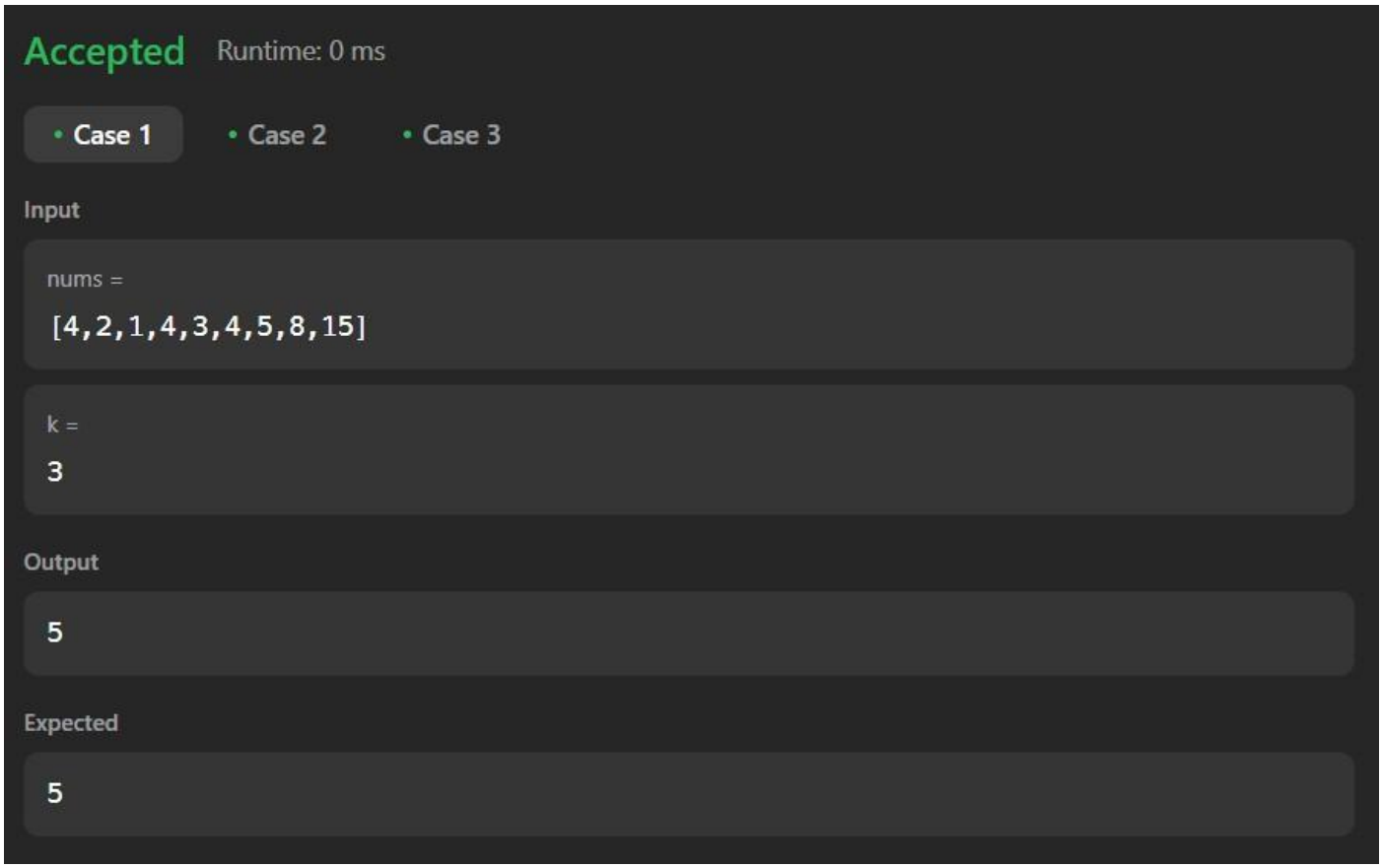
Problem 10.

```
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();
```

```
    }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[4,2,1,4,3,4,5,8,15]
k =
3
Output
5
Expected
5