Experiment 4

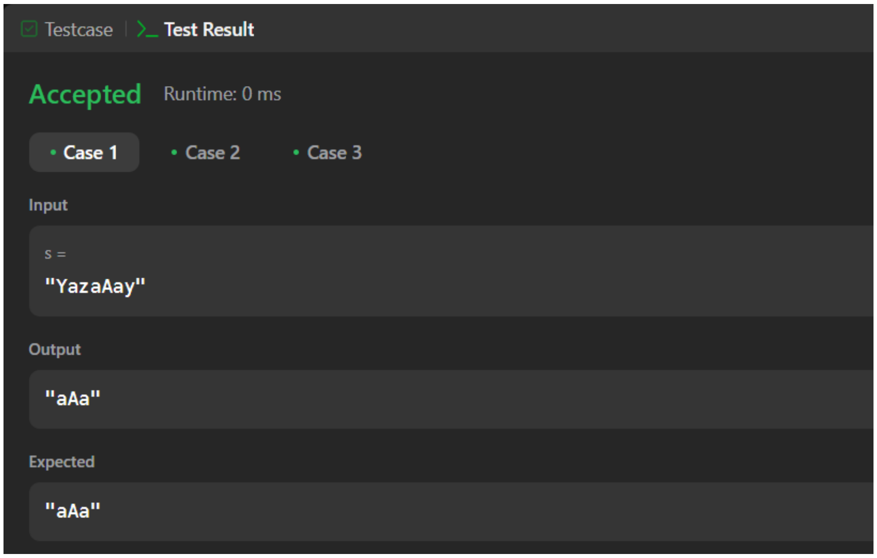
Name: Manik Naharia **UID:** 22BET10004
Branch: IT **Section:** 22BET_IOT-701/A
Semester: 6th **Date of Performance:** 14/02/25
Subject: Advanced Programming - 2 **Subject Code:** 22ITP-351

Problem 1. Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string.

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:



Accepted Runtime: 0 ms

Input

```
s =
"YazaAay"
```

Output

```
"aAa"
```

Expected

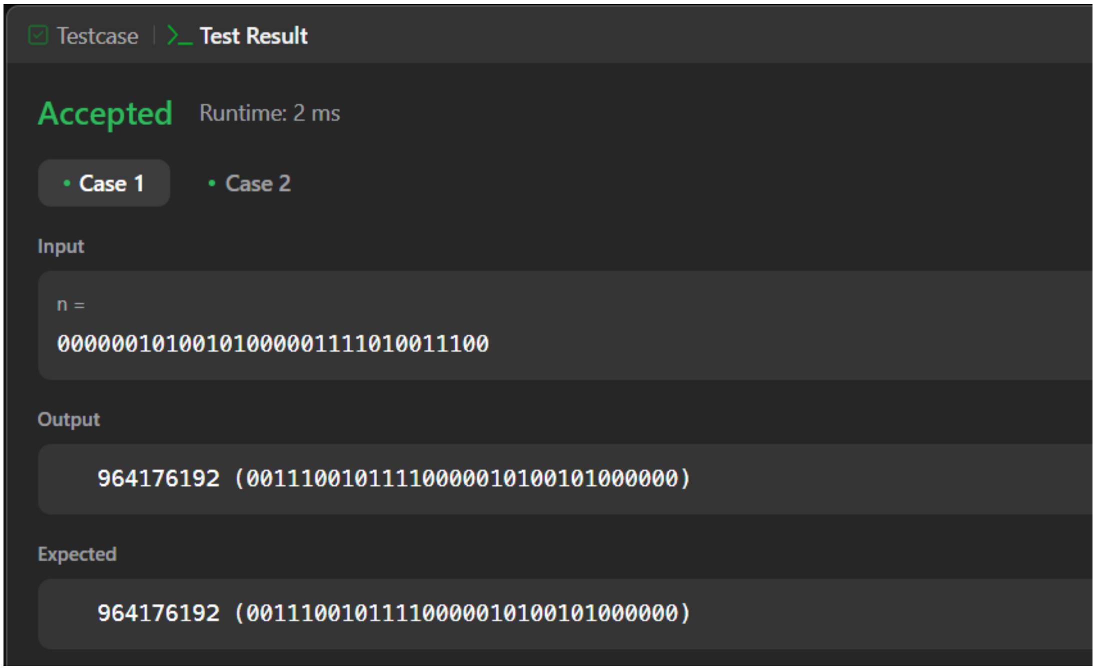
```
"aAa"
```

Problem 2. Reverse bits of a given 32 bits unsigned integer.

Code:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        n = (n >> 16) | (n << 16);
        n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);
        n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);
        n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);
        n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);
        return n;
    }
};
```

Output:



Accepted Runtime: 2 ms

Input

```
n =
00000010100101000001111010011100
```

Output

```
964176192 (00111001011110000010100101000000)
```

Expected

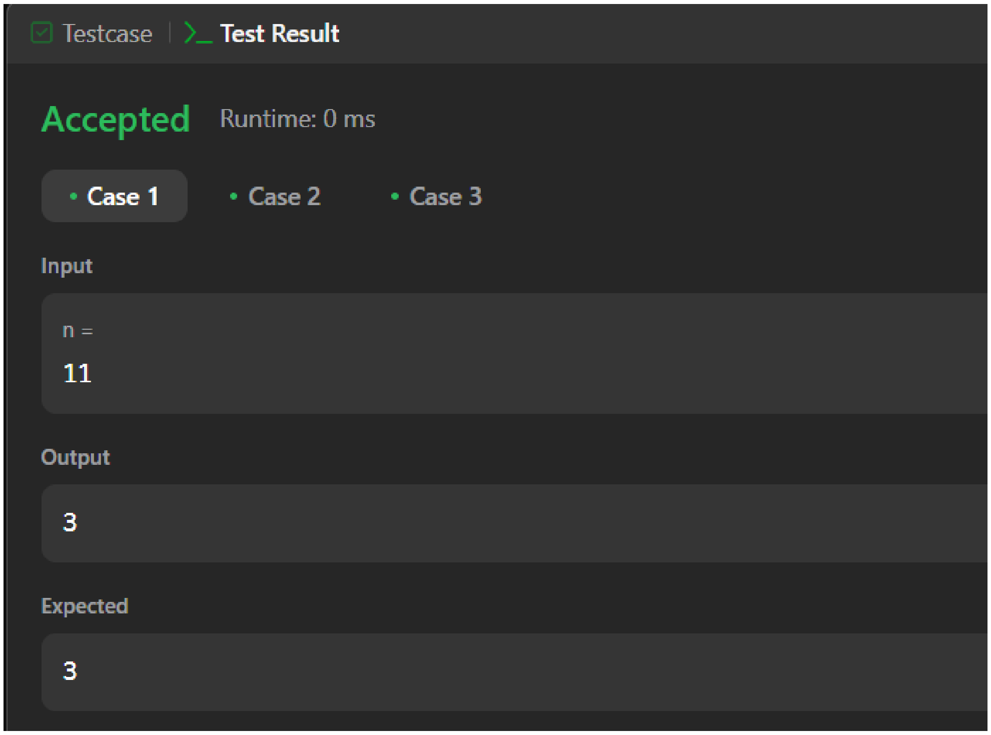
```
964176192 (00111001011110000010100101000000)
```

Problem 3. Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Code:

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

Output:



Accepted Runtime: 0 ms

Input

```
n =
11
```

Output

```
3
```

Expected

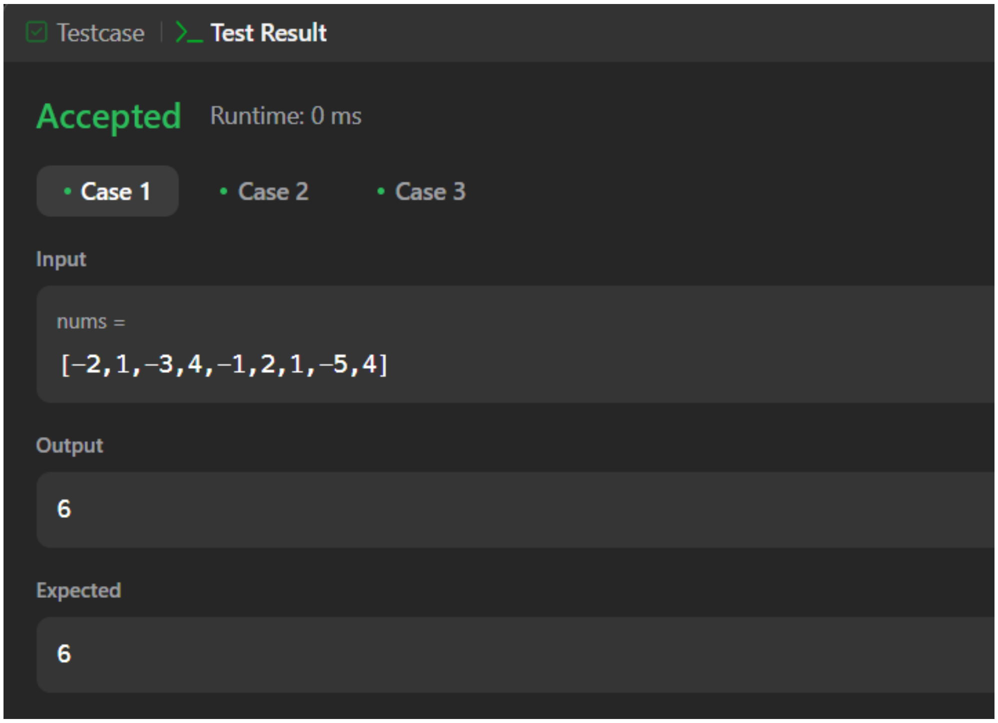
```
3
```

Problem 4. You Given an integer array nums, find the Subarray with the largest sum, and 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:



Accepted Runtime: 0 ms

Input

```
nums =
[-2,1,-3,4,-1,2,1,-5,4]
```

Output

```
6
```

Expected

```
6
```

Problem 5. Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

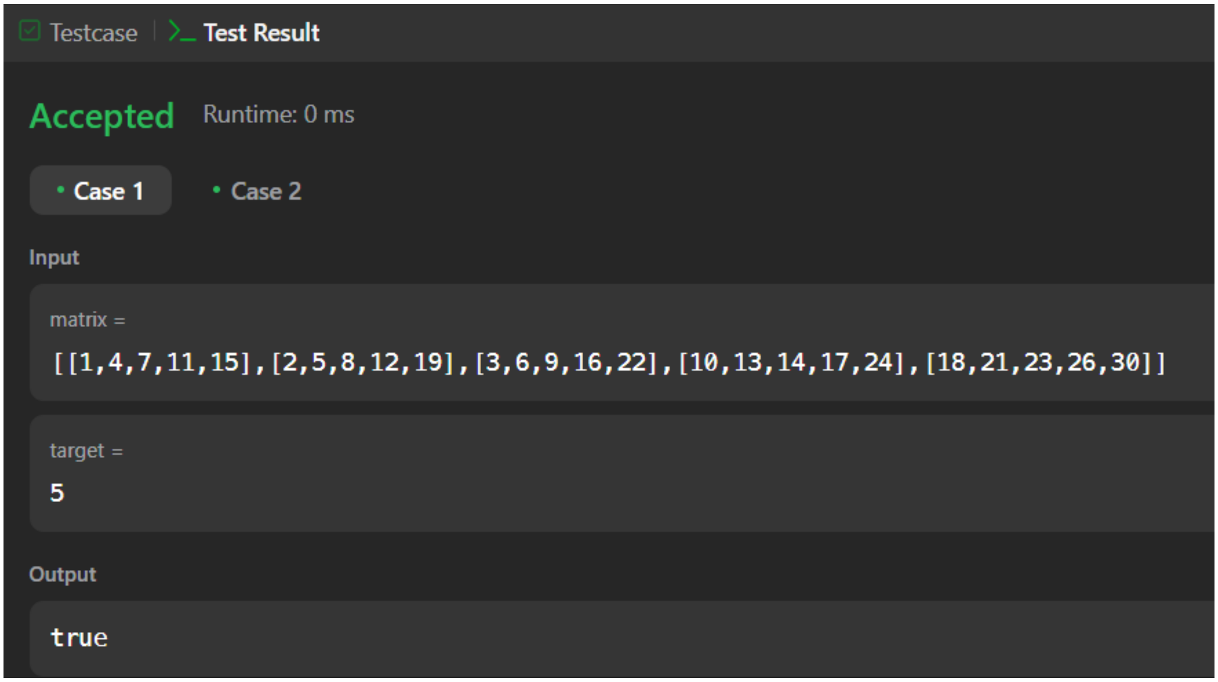
- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Code:

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



Accepted Runtime: 0 ms

Input

```
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =
5
```

Output

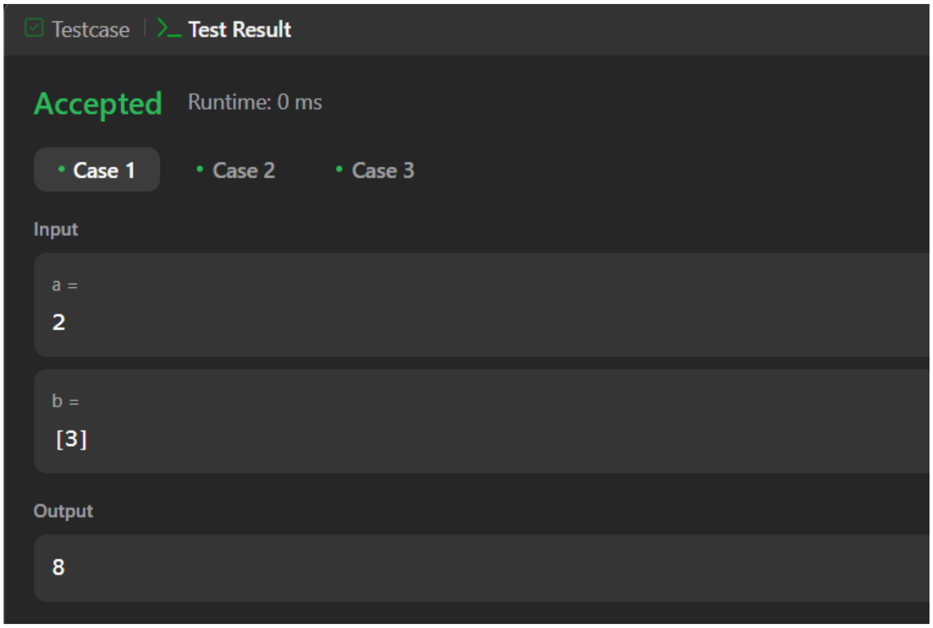
```
true
```

Problem 6. Your task is to calculate a^b 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 {
public:
   int binexp(long long a, long long b){
      a%=1337;
      int ans=1;
      while(b){
         if(b&1) ans=(ans*1ll*a)%1337;
         a=(a*1ll *a)%1337;
      b>>=1;
      }return ans;
   }
   int superPow(int a, vector<int>& b) {
     //ETF(1337)=1140
      long long sum=0;
      for(int i=0;i<b.size();++i){
         sum=((sum*10)+b[i])%1140;
      }
      return binexp(a,sum);
   }
};
```

Output:



Accepted Runtime: 0 ms

Input

a =
2

b =
[3]

Output

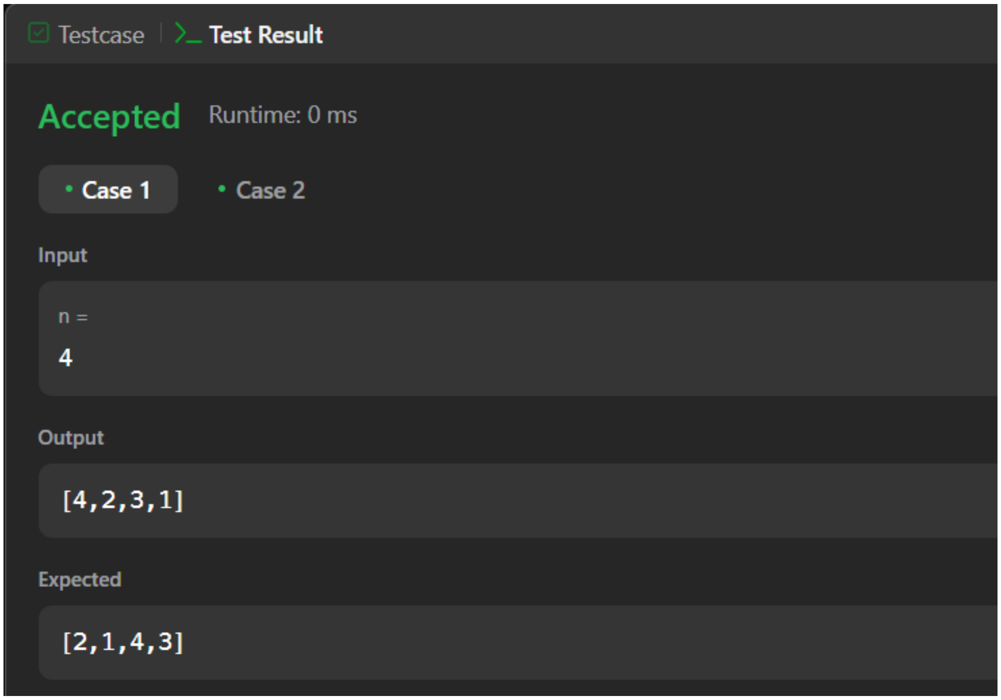
8

Problem 7. Given the integer n, return *any* ***beautiful*** *array* nums *of length* n. There will be at least one valid answer for the given n.

Code:

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if(n==1)
            return {1};
        vector<int> even = beautifulArray(n/2);
        vector<int> odd = beautifulArray(n-(n/2));
        vector<int>ans;
        for(auto e:even)
            ans.push_back(2*e);
        for(auto e:odd)
            ans.push_back((2*e)-1);
        return ans;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

```
n =
4
```

Output

```
[4,2,3,1]
```

Expected

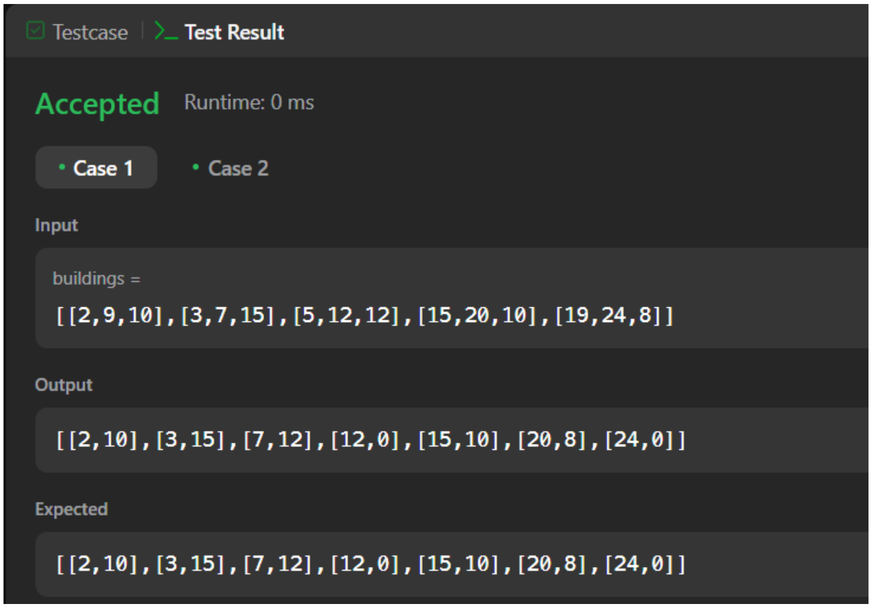
```
[2,1,4,3]
```

Problem 8. 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*.

Code:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> skyline;
        map<int, vector<pair<int, int>>> map; // key : pos, value : vector of <height, start|end> pairs
        for (auto& building : buildings) {
            map[building[0]].push_back({building[2], 0}); // add startpoint
            map[building[1]].push_back({building[2], 1}); // add endpoint
        }
        multiset<int> q;
        for (auto& [pos, heights] : map) {
            for (auto& [height, type] : heights) {
                if (type == 0) q.insert(height);
                else q.erase(q.find(height));
            }
            int newHeight = q.empty() ? 0 : *q.rbegin();
            if (!skyline.empty() && skyline.back()[1] == newHeight) continue;
            else skyline.push_back(vector<int>({pos, newHeight}));
        }
        return skyline;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

buildings =

```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Output

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Expected

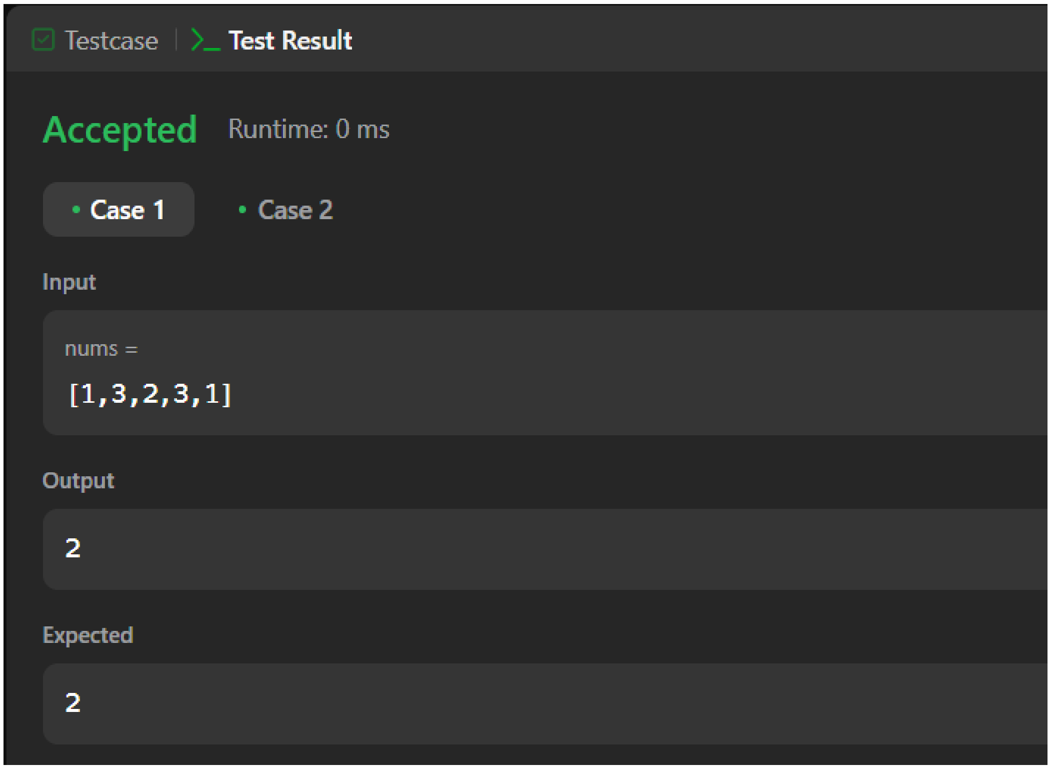
```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Problem 9. 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 = 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:



Accepted Runtime: 0 ms

Input

nums =

```
[1,3,2,3,1]
```

Output

```
2
```

Expected

```
2
```

Problem 10. You are given an integer array nums and an integer k.

Find the longest subsequence of nums that meets the following requirements:

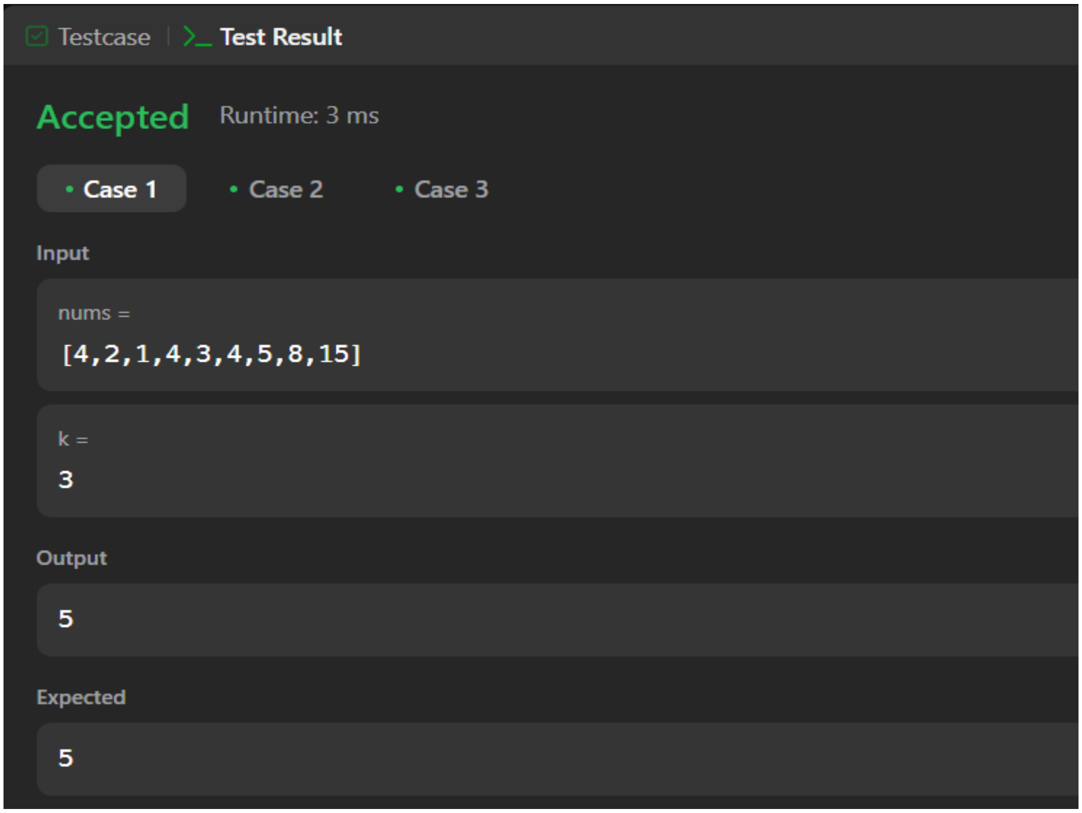
- The subsequence is strictly increasing and
- The difference between adjacent elements in the subsequence is at most k.

Return *the length of the longest subsequence that meets the requirements.*

Code:

```
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);
            upd(nums[i], q + 1, 0, 0, x - 1);
        }
        return res;
    }
};
```

Output:



Accepted Runtime: 3 ms

Input

```
nums =
[4,2,1,4,3,4,5,8,15]
```

```
k =
3
```

Output

```
5
```

Expected

```
5
```