# Problem 1

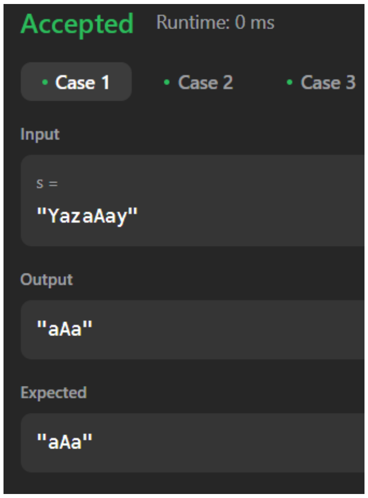
## Aim:

Longest Nice Substring

## Code:

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        string output = "";
        int count = 0;
        for(int i = 0;i<s.length();i++){
            int smallMask=0;
            int largeMask = 0;
            char ch = s[i];
            int chint = 0;
            if(ch>=65 && ch<=90){
                chint = ch-'A';
                largeMask = 1<<chint;
            }
            else{
                chint = ch-'a';
                smallMask = 1<<chint;
            }
            for(int j = i+1;j<s.length();j++){
                ch = s[j];
                if(ch>=65 && ch<=90){
                    chint = ch-'A';
                    largeMask |= 1<<chint;
                }
                else{
                    chint = ch-'a';
                    smallMask |= 1<<chint;
                }
                //checking for nice
                if((smallMask^largeMask) == 0){
                    if(count<j-i+1){
                        count = j-i+1;
                        string temp(s.begin()+i,s.begin()+j+1);
                        output = temp;
                    }
                }
            }
        }
        return output;
    }
};
```

## Output:



Accepted Runtime: 0 ms

Input

s =

```
"YazaAay"
```

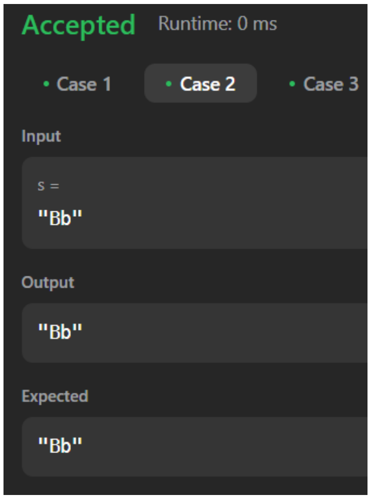
Output

```
"aAa"
```

Expected

```
"aAa"
```

Case 1



Accepted Runtime: 0 ms

Input

s =

```
"Bb"
```

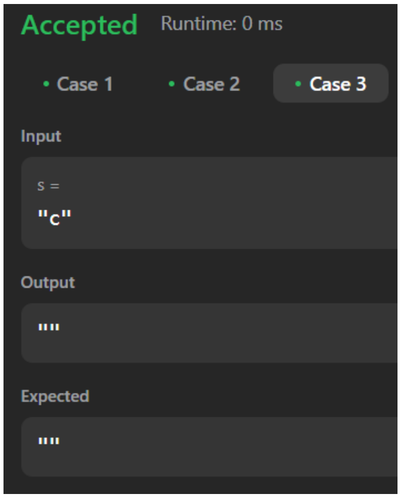
Output

```
"Bb"
```

Expected

```
"Bb"
```

Case 2



Accepted Runtime: 0 ms

Input

s =

```
"c"
```

Output

```
""
```

Expected

```
""
```

Case 3

## **Problem 2**

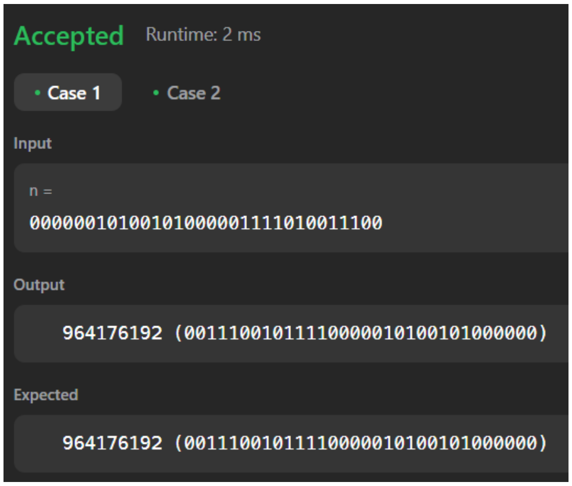
**Aim:**

Reverse Bits

## Code:

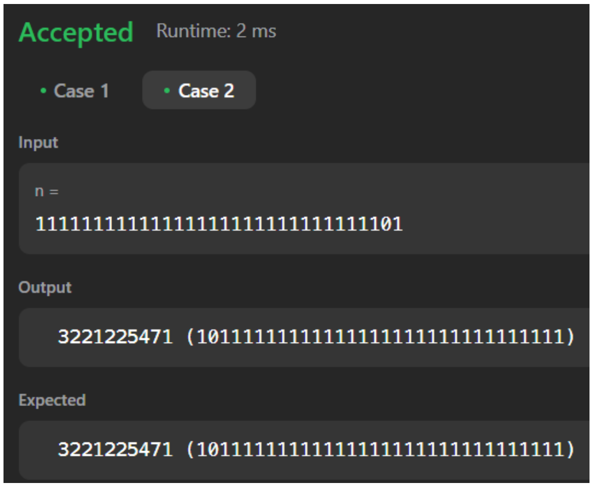
```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            int bit = n & 1;        // Extract the least significant bit
            result = (result << 1) | bit; // Append the bit to the result
            n = n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

## Output:



```
Input
n =
00000010100101000001111010011100
Output
964176192 (00111001011110000010100101000000)
Expected
964176192 (00111001011110000010100101000000)
```

Test Case 1



```
Input
n =
11111111111111111111111111111101
Output
3221225471 (10111111111111111111111111111111)
Expected
3221225471 (10111111111111111111111111111111)
```

Test Case 2

## **Problem 3**

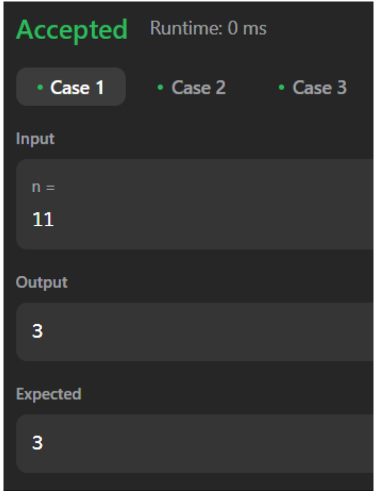
## Aim:

Number of 1 bits

**Code:**

```
class Solution {
public:
    int hammingWeight(int n) {
       bitset<32> b(n);
       int ans=0;
       for(size_t i=0;i<b.size();i++){
        if(b[i]==1) ans++;
       }
       return ans;
    }
};
```

**Output:**



Input

n =
11

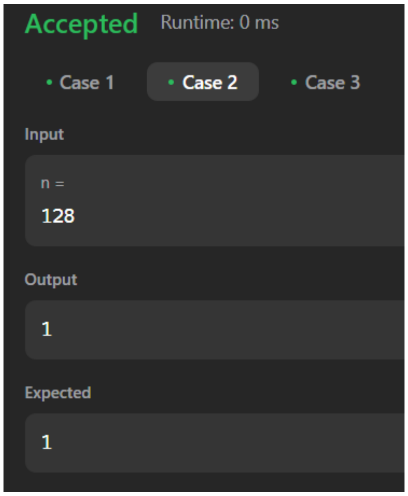
Output

3

Expected

3

Case 1



Input

n =
128

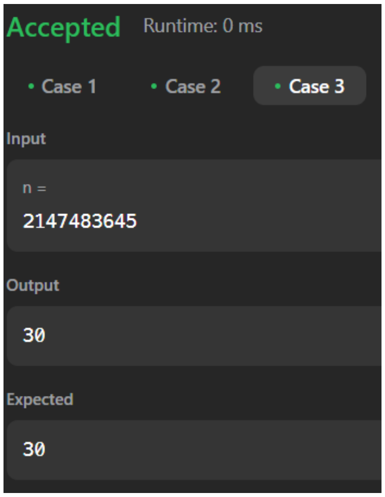
Output

1

Expected

1

Case 2



Input

n =
2147483645

Output

30

Expected

30

Case 3

## Problem 4

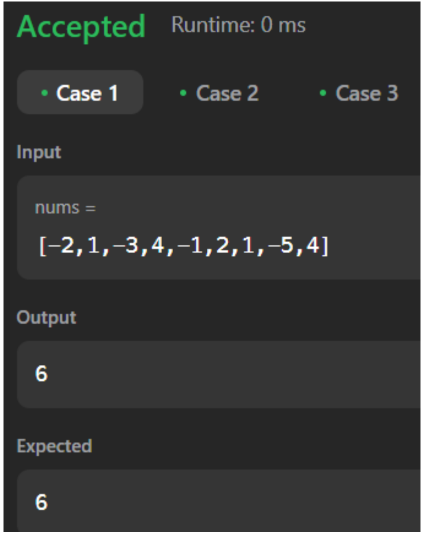
**Aim:**

Max Subarray

**Code:**

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

        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }

            total += n;
            res = max(res, total);
        }

        return res;
    }
};
```

**Output:**



Input

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

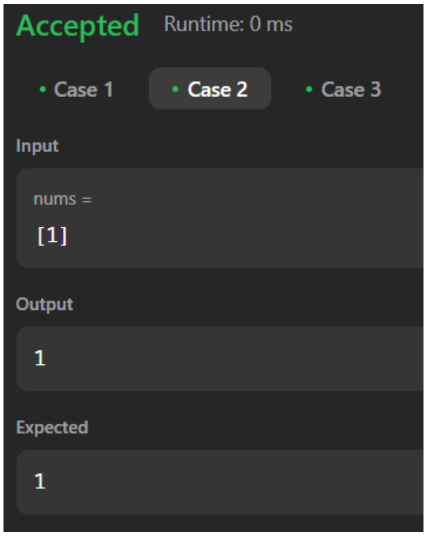
Output

```
6
```

Expected

```
6
```

Case 1



Input

```
nums =
[1]
```

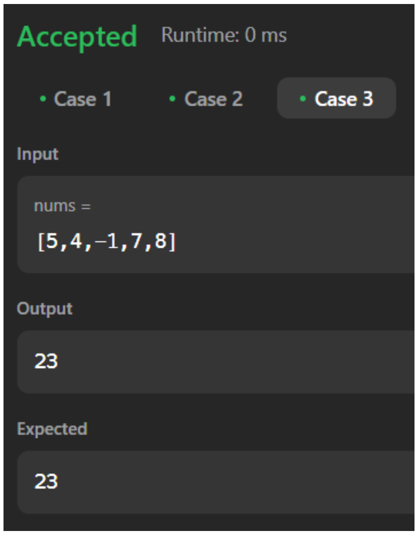
Output

```
1
```

Expected

```
1
```

Case 2



Input

```
nums =
[5,4,-1,7,8]
```

Output

```
23
```

Expected

```
23
```

Case 3

## Problem 5

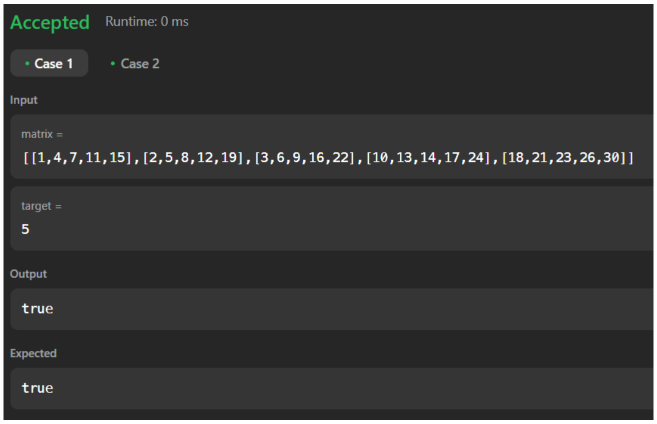
**Aim:**

Search 2d matrix 2

## Code:

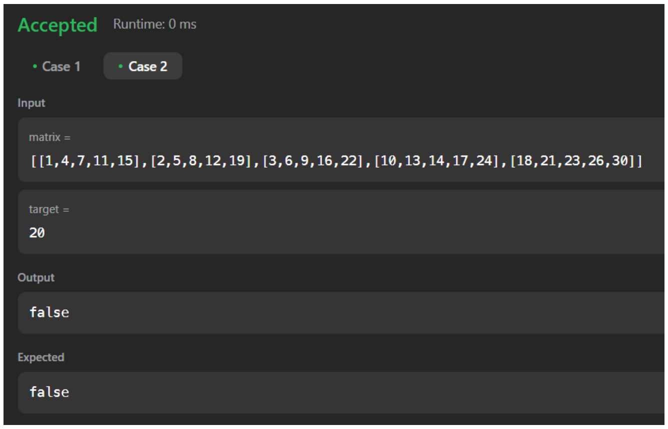
```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int n = matrix.size(), m = matrix[0].size();
        int row = 0, col = m - 1;

        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
• Case 1   • Case 2
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
Expected
true
```

Case 1



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
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 =
20
Output
false
Expected
false
```

Case 2

# **Problem 6**

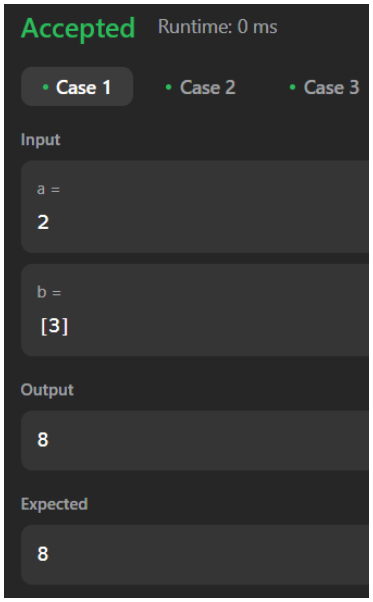
## Aim:

Super Pow

**Code:**

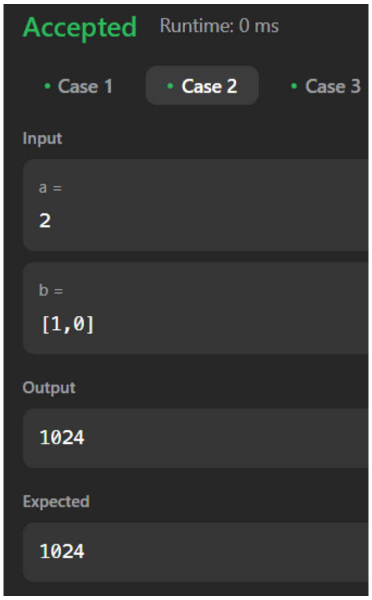
```
class Solution {
private:
    int solve(int base, int power, int mod) {
        int ans = 1;
        while (power > 0) {
            if (power & 1) {
                ans = (ans * base) % mod;
            }
            base = (base * base) % mod;
            power >>= 1;
        }
        return ans;
    }

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);
    }
};
```

**Output:**



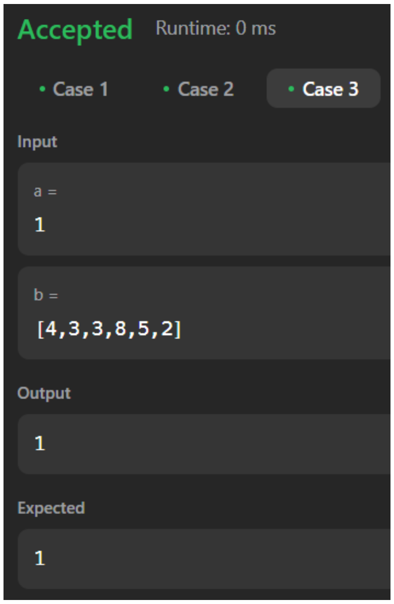
```
Input
a =
2
b =
[3]
Output
8
Expected
8
```

Case 1



```
Input
a =
2
b =
[1,0]
Output
1024
Expected
1024
```

Case 2



```
Input
a =
1
b =
[4,3,3,8,5,2]
Output
1
Expected
1
```

Case 3

## **Problem 7**

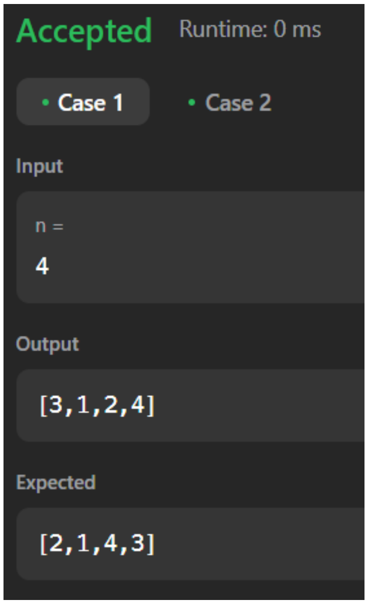
**Aim:**

Beautiful Array

**Code:**

```
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:**



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n =

4

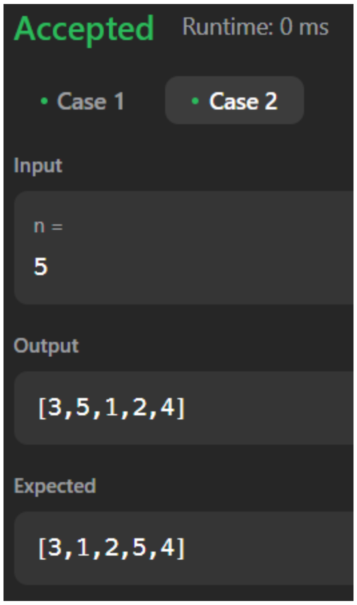
Output

[3,1,2,4]

Expected

[2,1,4,3]

Case 1



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n =

5

Output

[3,5,1,2,4]

Expected

[3,1,2,5,4]

Case 2

## Problem 8

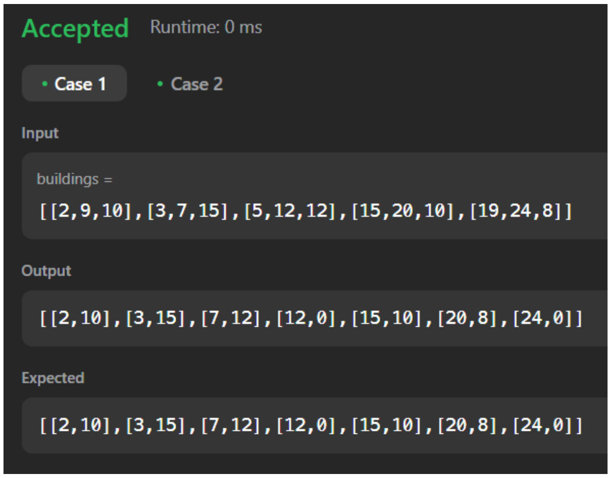
**Aim:**

The Skyline Problem

## Code:

```
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;
    }
};
```

## Output:



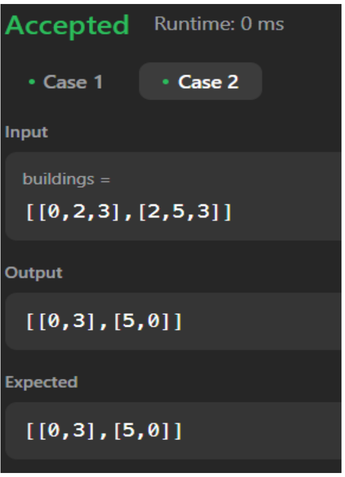
Input

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]]
```

Case 1



Input

buildings =
```
[[0,2,3],[2,5,3]]
```

Output
```
[[0,3],[5,0]]
```

Expected
```
[[0,3],[5,0]]
```

Case 2

## **Problem 9**

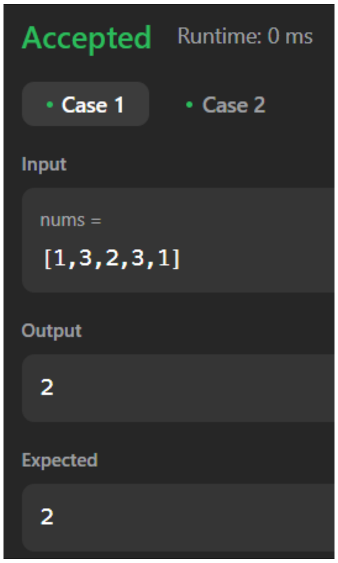
## Aim:

Reverse Pairs

## Code:

```
class SegTree {
private:
    int tree_size;
    vector<int> tree;

    void update(int lx, int rx, int ni, int idx) {

        if (rx - lx == 1) {
            tree[ni]++;
            return;
        }

        int m = (lx + rx) >> 1;

        if (idx < m)
            update(lx, m, ni * 2 + 1, idx);
        else
            update(m, rx, ni * 2 + 2, idx);

        tree[ni] = tree[ni * 2 + 1] + tree[ni * 2 + 2];
    }

    int query(int l, int r, int lx, int rx, int ni) {

        if (l >= rx || r <= lx)
            return 0;

        if (l <= lx && r >= rx)
            return tree[ni];

        int m = (lx + rx) >> 1;
        return query(l, r, lx, m, ni * 2 + 1) + query(l, r, m, rx, ni * 2 + 2);
    }

public:
    SegTree(int n) {
        tree_size = 1;

        while (tree_size < n)
            tree_size <<= 1;

        tree = vector<int>(tree_size * 2);
    }

    void update(int idx) {
        update(0, tree_size, 0, idx);
    }

    int query(int l, int r) {
        return query(l, r + 1, 0, tree_size, 0);
    }
};

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

        int n = nums.size();
        set<long long> values;

        for(const auto& num : nums) {
            values.insert(num);
            values.insert(2LL * num);
        }

        int last_index = 0;
        unordered_map<long long, int> values_indices;

        for(const auto& val : values)
            values_indices[val] = last_index++;

        SegTree seg_tree(last_index);
        int ans = 0;

        for(int i = 0; i < n; ++i) {
            ans += seg_tree.query(values_indices[2LL * nums[i]] + 1, last_index);
            seg_tree.update(values_indices[nums[i]]);
        }

        return ans;
    }
};
```

**Output:**



Accepted Runtime: 0 ms

Input

nums =

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

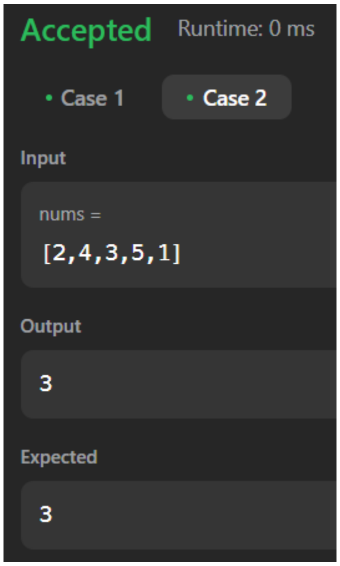
Output

```
2
```

Expected

```
2
```

Case 1



Accepted Runtime: 0 ms

Input

nums =

```
[2,4,3,5,1]
```

Output

```
3
```

Expected

```
3
```

Case 2

## <u>Problem 10</u>

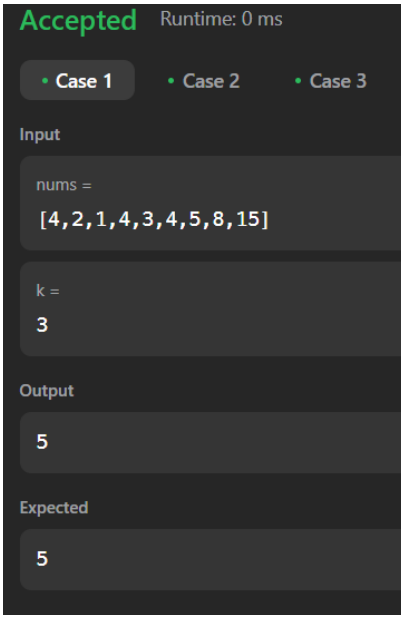
## Aim:

Longest increasing subsequence 2

## Code:

```
class Solution {
public:
    vector<int>tree;
    void update(int node,int st,int end,int i,int val){
        if(st==end){
            tree[node]=max(tree[node],val);
            return;
        }
        int mid=(st+end)/2;
        if(i<=mid){
            update(node*2,st,mid,i,val);
        }else{
            update(node*2+1,mid+1,end,i,val);
        }
        tree[node]=max(tree[node*2],tree[node*2+1]);
    }
    int query(int node,int st,int end,int x,int y){
        if(x>end || y<st) return -1e9;
        if(st>=x && end<=y){
            return tree[node];
        }
        int mid=(st+end)/2;
        int left=query(2*node,st,mid,x,y);
        int right=query(2*node+1,mid+1,end,x,y);
        return max(left,right);
    }
    int lengthOfLIS(vector<int>& nums, int k) {
        int n=nums.size();
        if(n==1) return 1;
        int m=*max_element(nums.begin(),nums.end());
        tree.clear();
        tree.resize(4*m+10);
        for(int i=n-1;i>=0;i--){
            int l=nums[i]+1,r=min(nums[i]+k,m);
            int x=query(1,0,m,l,r);
            if(x==-1e9) x=0;
            update(1,0,m,nums[i],x+1);
        }
        return tree[1];
    }
};
```

## Output:



Accepted Runtime: 0 ms

Input

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

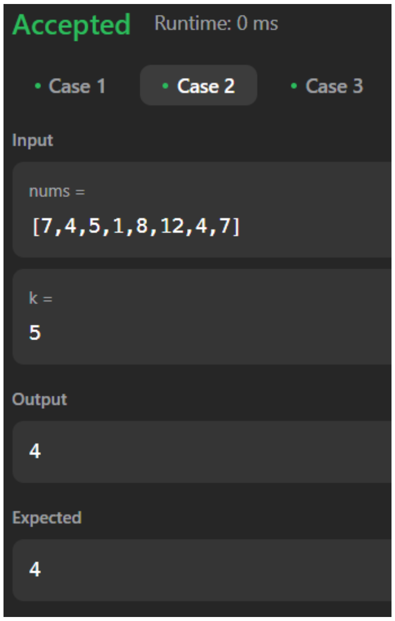
Output

```
5
```

Expected

```
5
```

Case 1



Accepted Runtime: 0 ms

Input

```
nums =
[7,4,5,1,8,12,4,7]
k =
5
```

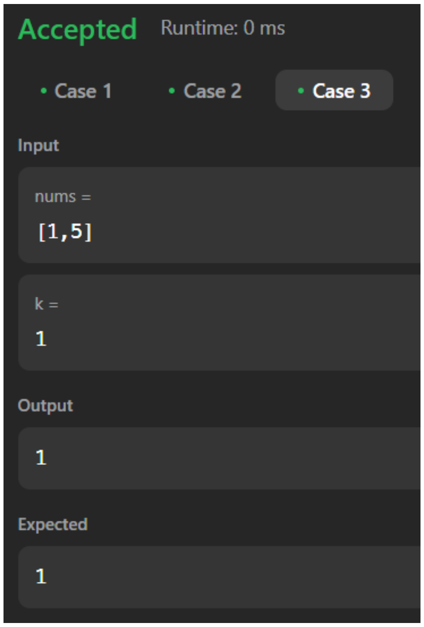
Output

```
4
```

Expected

```
4
```

Case 2



Accepted Runtime: 0 ms

Input

```
nums =
[1,5]
k =
1
```

Output

```
1
```

Expected

```
1
```

Case 3