CHANDIGARH UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment 4

Student Name: Shivansh Singh **UID: 22BET10105**

Branch: IT **Section/Group: 22BET_IOT-701/A**

Semester: 6th **Date of Performance:14.02.25**

Subject Name: AP Lab - 2 **Subject Code: 22ITP-351**

1. **Aim:** To improve problem-solving skills by solving diverse LeetCode problems, covering topics such as bit manipulation, dynamic programming, divide and conquer, binary search, and advanced data structures.
   i.) Number of 1 bits
   ii.) Max Subarray
   iii.) Search 2d matrix 2
   iv.) Super Pow
   v.) Beautiful Array
   vi.) The Skyline Problem
   vii.) Reverse Pairs

2. Objective:

- Learn and apply different algorithmic strategies.
- Strengthen skills in bit manipulation and dynamic programming.
- Tackle problems using binary search and divide & conquer techniques.
- Deepen understanding of advanced data structures.
- Focus on writing optimized and high-performance code.
- Improve logical reasoning and debugging abilities.
- Gain practical experience by solving problems on LeetCode.

3. Code:

Problem 1: Number of 1 bits

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

Problem 2: Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int max_sum=nums[0];
        int curr_sum=nums[0];
        for(int i=1;i<nums.size();i++){
            curr_sum=max(nums[i],curr_sum+nums[i]);
            max_sum=max(curr_sum,max_sum);
        }
        return max_sum;
    }
};
```

Problem 3: Search a 2D Matrix II

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
         int m = matrix.size();
        if (m == 0) return false;
        int n = matrix[0].size();
        // Start from the top-right corner
        int row = 0, col = n - 1;
        while (row < m && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;  // Move left
            } else {
                row++;  // Move down
            }
        }
        return false;
    }
};
```

Problem 4: Super Pow

```
class Solution {
public:
const int MOD = 1337;

    // Function to calculate (x^y) % mod using modular exponentiation
    int powerMod(int x, int y, int mod) {
        int result = 1;
        x %= mod;  // Reduce base modulo

        while (y > 0) {
            if (y % 2 == 1) {  // If y is odd, multiply x with the result
                result = (result * x) % mod;
            }
            x = (x * x) % mod;  // Square x
            y /= 2;  // Reduce exponent
        }
        return result;
    }
    int superPow(int a, vector<int>& b) {
        a %= MOD;  // Reduce base modulo 1337
        int result = 1;

        for (int digit : b) {
            // result = (result^10 * a^digit) % 1337
            result = powerMod(result, 10, MOD) * powerMod(a, digit, MOD) % MOD;
        }
        return result;
    }
};
```

Problem 5: Beautiful Array

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
       vector<int> result = {1}; // Base case: beautiful array of length 1
        while (result.size() < n) {
            vector<int> next;
            // Construct odd numbers (2 * x - 1)
            for (int x : result) {
                if (2 * x - 1 <= n) {
                    next.push_back(2 * x - 1);
                }
            }
            // Construct even numbers (2 * x)
            for (int x : result) {
                if (2 * x <= n) {
                    next.push_back(2 * x);
                }
            }
            result = next; // Update the result
        }
        return result;
    }
};
```

Problem 6: The Skyline Birthday

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);  // Start of a building (negative height)
            events.emplace_back(b[1], b[2]);   // End of a building (positive height)
        }
        // Sort events: If x-coordinates are the same, process by height
        sort(events.begin(), events.end());
        multiset<int> heights = {0}; // Max heap using multiset (auto-sorted)
        vector<vector<int>> skyline;
        int prevMax = 0;
        for (auto& [x, h] : events) {
            if (h < 0) {
                heights.insert(-h);  // Add new building height
            } else {
                heights.erase(heights.find(h));  // Remove building height
            }
            int currMax = *heights.rbegin();  // Get current max height
            if (currMax != prevMax) {  // If height changes, add a key point
                skyline.push_back({x, currMax});
                prevMax = currMax;
            }
        }
        return skyline;
    }
};
```

Problem 7: Reverse Pair

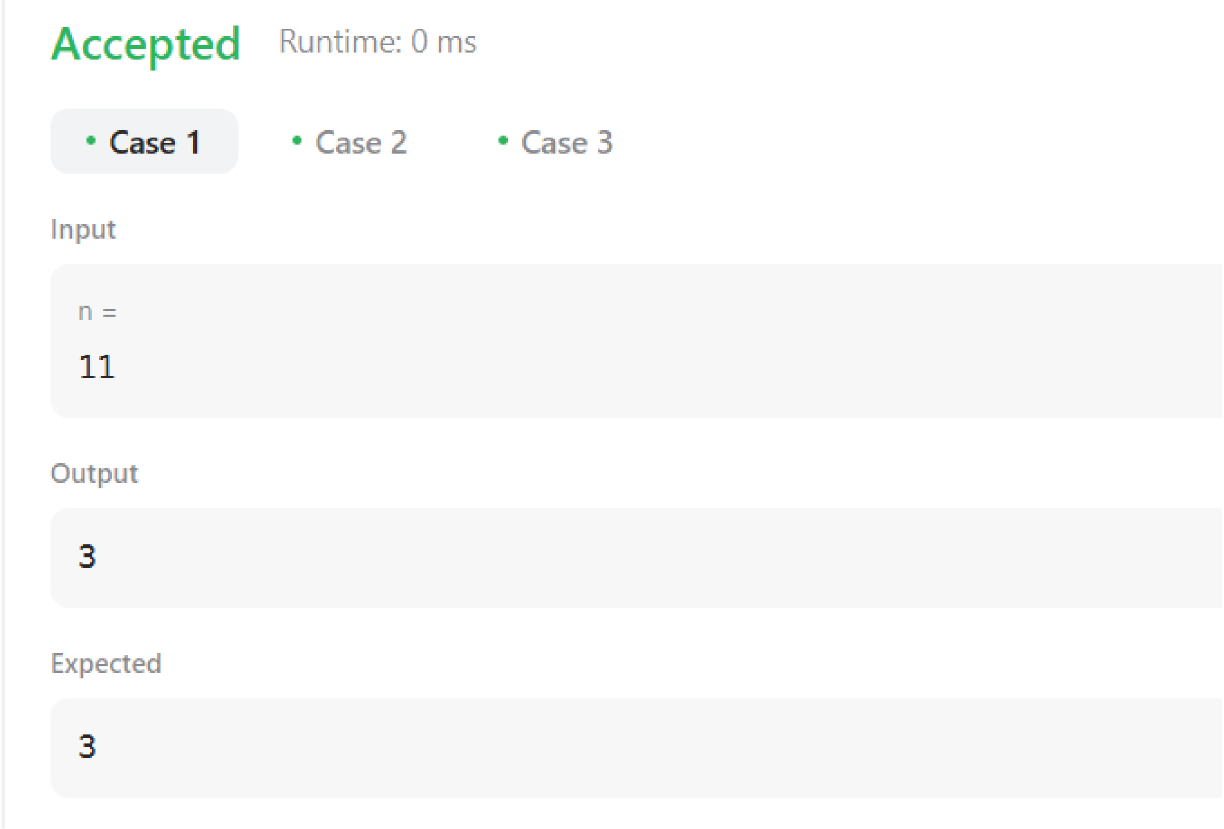
```
class Solution {
public:
    int mergeAndCount(vector<int>& nums, int left, int mid, int right) {
        int count = 0, j = mid + 1;
        // Count reverse pairs
        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2LL * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }
        // Merge two sorted halves
        vector<int> temp;
        int i = left, k = mid + 1;
        while (i <= mid && k <= right) {
            if (nums[i] <= nums[k]) {
                temp.push_back(nums[i++]);
            } else {
                temp.push_back(nums[k++]);
            }
        }
        while (i <= mid) temp.push_back(nums[i++]);
        while (k <= right) temp.push_back(nums[k++]);

        // Copy sorted array back
        for (int i = left; i <= right; i++) {
            nums[i] = temp[i - left];
        }

        return count;
    }
    int mergeSortAndCount(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSortAndCount(nums, left, mid) +
                    mergeSortAndCount(nums, mid + 1, right) +
                    mergeAndCount(nums, left, mid, right);
        return count;
    }

    int reversePairs(vector<int>& nums) {  // <-- Only declared once
        return mergeSortAndCount(nums, 0, nums.size() - 1);
    }
};
```

4. Output:



Accepted Runtime: 0 ms

Input

n =

11

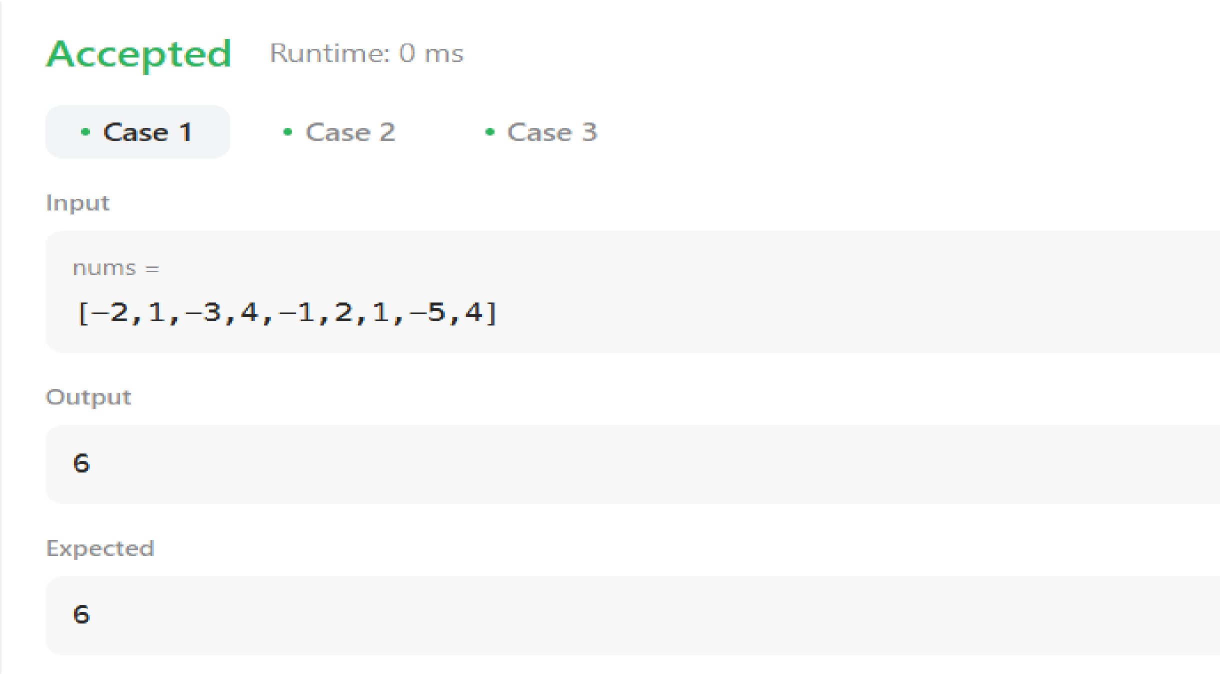
Output

3

Expected

3

Fig 1. Number of 1 bits



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =

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

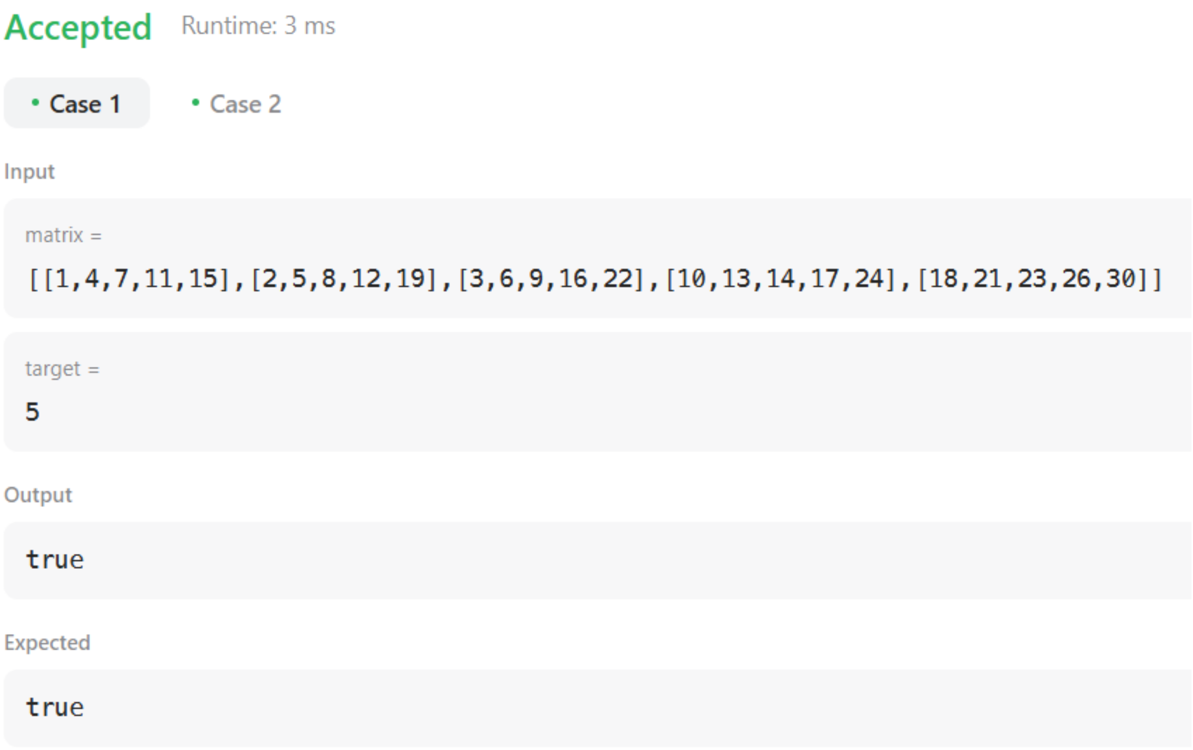
Output

```
6
```

Expected

```
6
```

Fig 2. Maximum Subarray



Accepted Runtime: 3 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
```

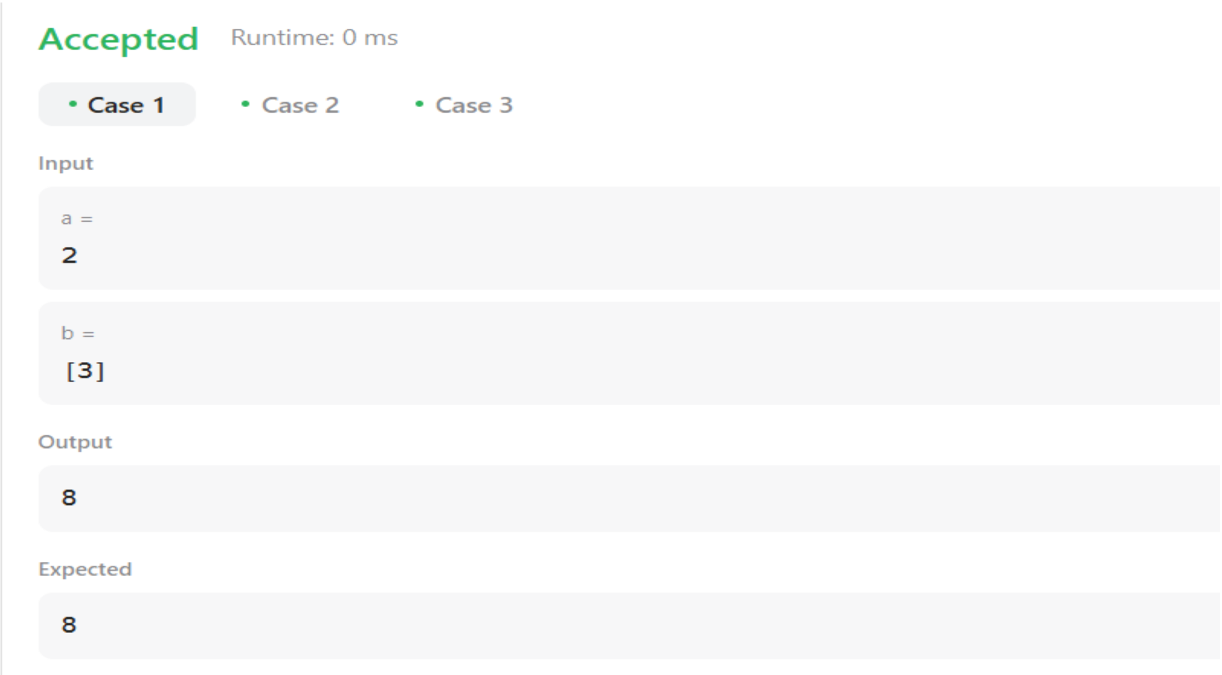
Output

```
true
```

Expected

```
true
```

Fig 3. Search a 2D matrix



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

a =

2

b =

[3]

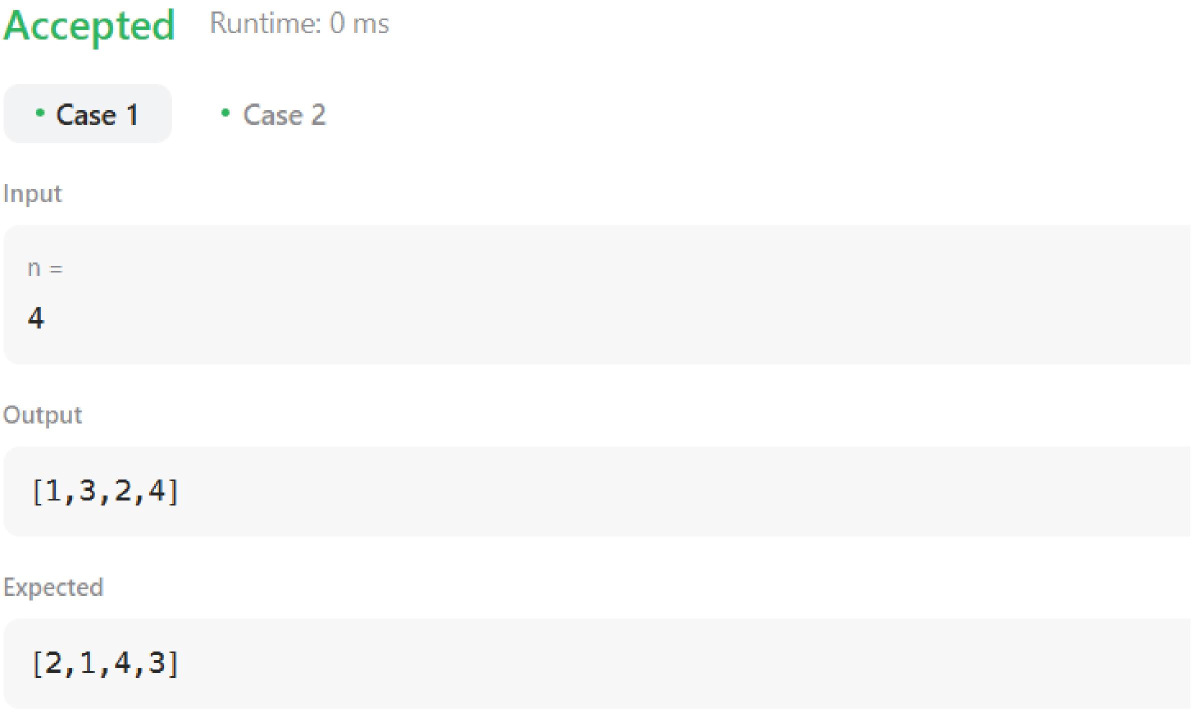
Output

8

Expected

8

Fig 4. Super Pow



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n =

4

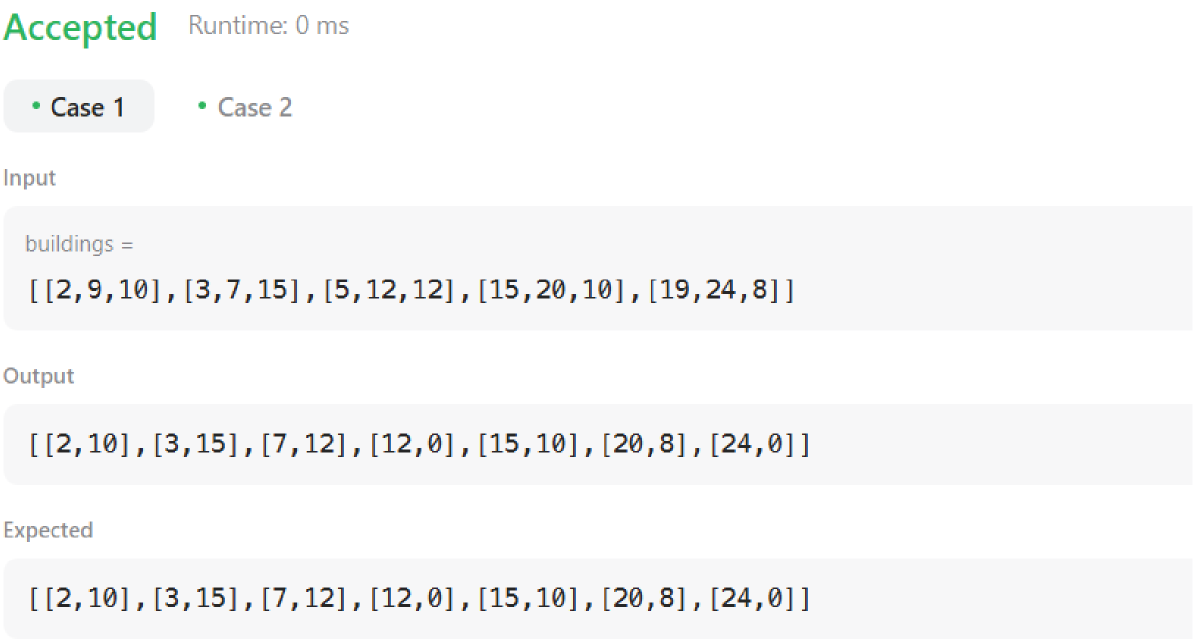
Output

[1,3,2,4]

Expected

[2,1,4,3]

Fig 5. Beautiful Array



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

buildings =

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

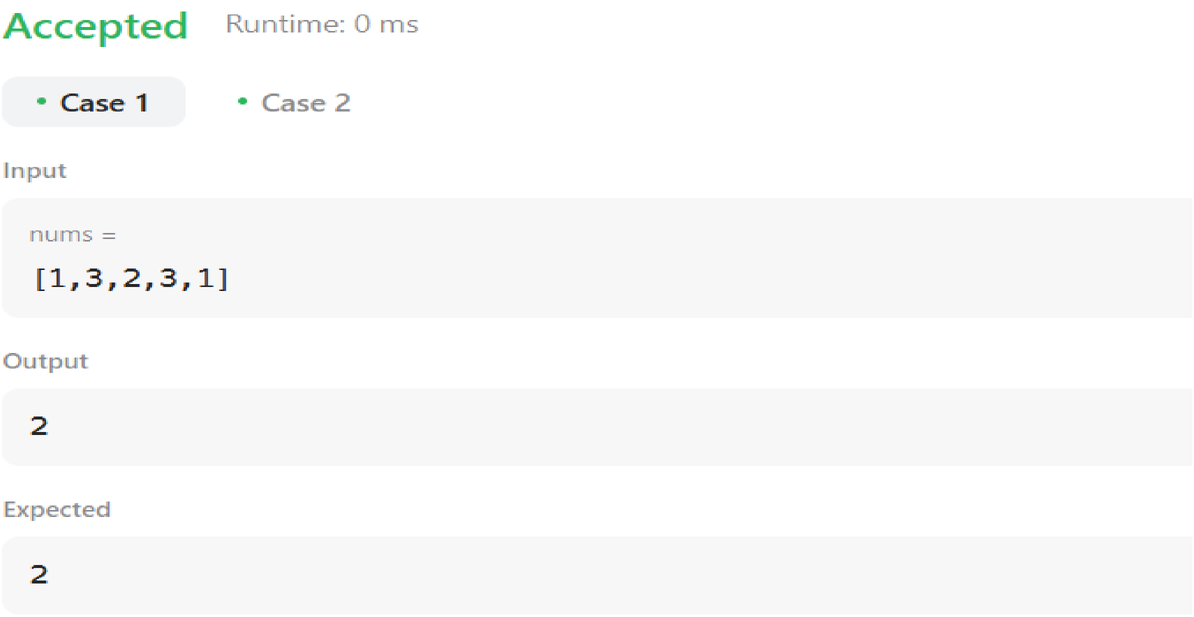
Output

```
[[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]]
```

Fig 6. The Skyline problem



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =

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

Output

```
2
```

Expected

```
2
```

Fig 7. Reverse Pairs

5. Learning Outcomes:

- Gained expertise in linked lists and advanced data structures, including Fenwick Trees, Segment Trees, and Heaps.
- Enhanced proficiency in searching and sorting algorithms, utilizing binary search and merge sort-based counting techniques.
- Developed a strong understanding of bit manipulation for optimizing operations on binary representations.
- Applied modular arithmetic principles to efficiently manage large computations in problems like **Super Pow**.
- Tackled intricate computational geometry challenges using priority queues and sweep line algorithms.