Experiment 4

Student Name: Ronit Jain **UID: 22BET10242**

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) Longest Nice Substring
   ii) Reverse Bits
   iii) Number of 1 bits
   iv) Max Subarray
   v) Search 2d matrix 2
   vi) Super Pow
   vii) Beautiful Array

2. **Objective:**
   - Understand and implement various algorithmic techniques.
   - Improve proficiency in bit manipulation and dynamic programming.
   - Solve problems involving binary search and divide & conquer.
   - Enhance knowledge of advanced data structures.
   - Optimize code for efficiency and performance.
   - Strengthen logical reasoning and debugging skills.
   - Gain hands-on experience with problem-solving on LeetCode.

3. **Code:**

Problem 1: Longest Nice Substring

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        if (s.size() < 2) return "";
        unordered_set<char> charSet(s.begin(), s.end());
```

```
        for (int i = 0; i < s.size(); i++) {
            char c = s[i];
            if (charSet.count(tolower(c)) && charSet.count(toupper(c))) {
                continue;
            }
            // Split the string at the first character that breaks the nice condition
            string left = longestNiceSubstring(s.substr(0, i));
            string right = longestNiceSubstring(s.substr(i + 1));
            return left.size() >= right.size() ? left : right;
        }
        return s;
    }
};
```

Problem 2: Reverse Bits

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

Problem 3: Number of 1 bits

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

Problem 4: Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxi = INT_MIN;
        int sum = 0;

        for(int i = 0; i<nums.size(); i++){
            sum += nums[i];
        if(sum > maxi){
            maxi = sum;
        }
        if(sum < 0){
            sum = 0;
        }
        }
        return maxi;
    }
};
```

Problem 5: 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 6: 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 7: 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 8: 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 9: 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:

Fig 1. Longest Substring

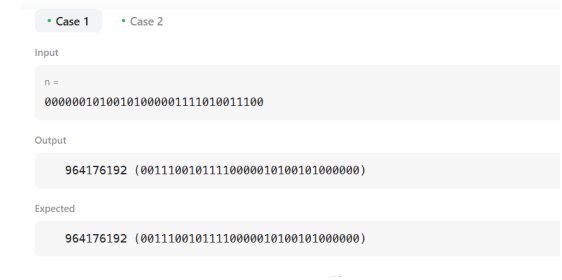

Fig 2. Reverse Bits

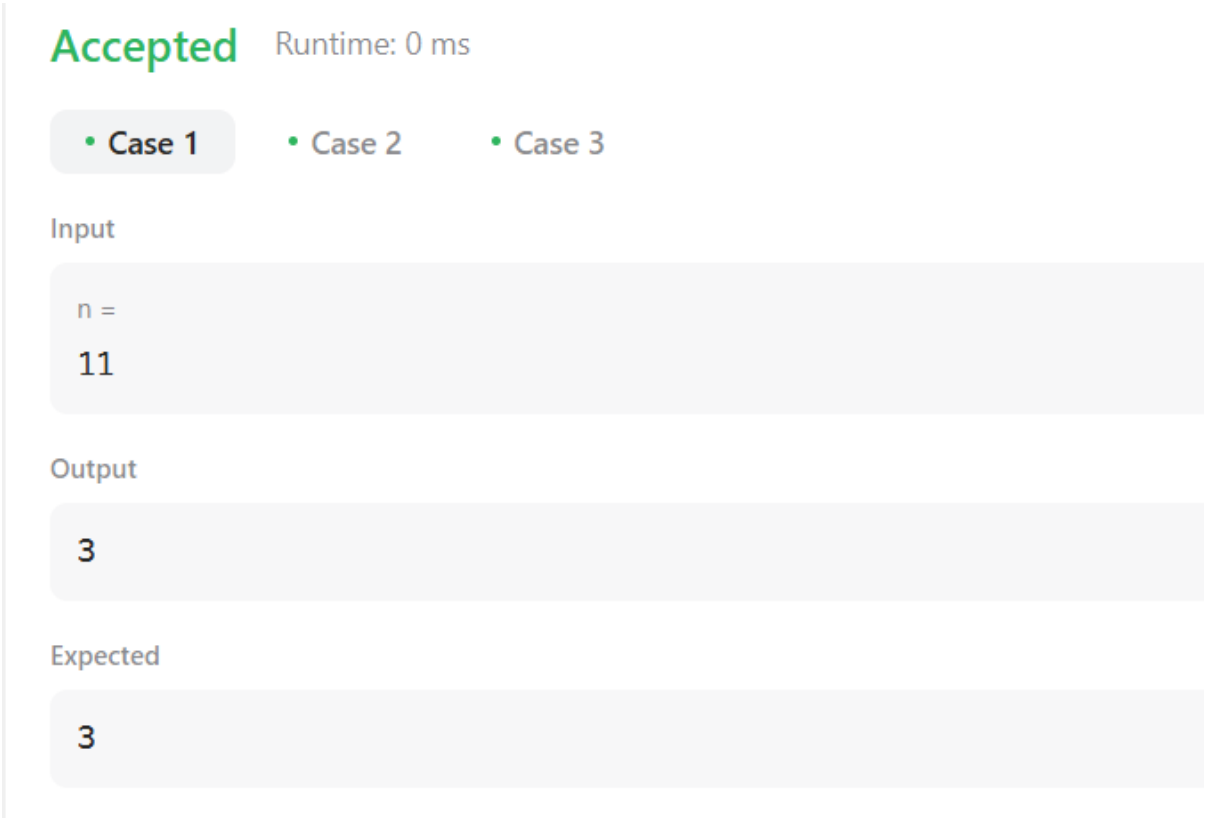

Fig 3. Number of 1 bits

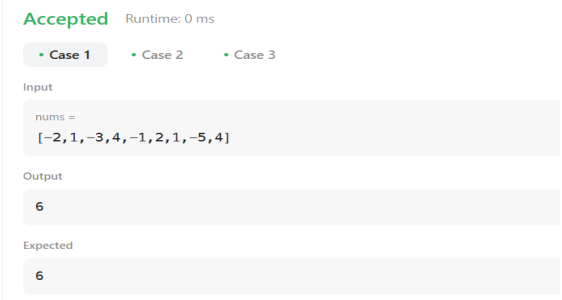

Fig 4. Maximum Subarray

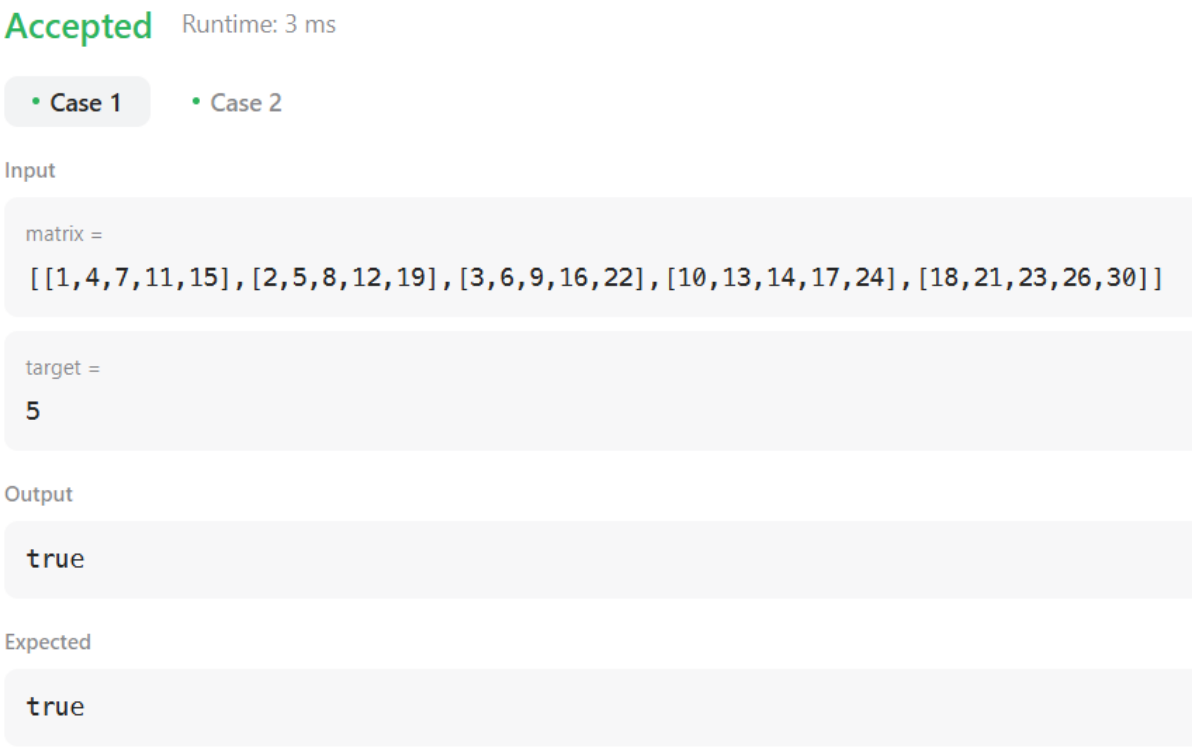

Fig 5. Search a 2D matrix

Fig 6. Super Pow

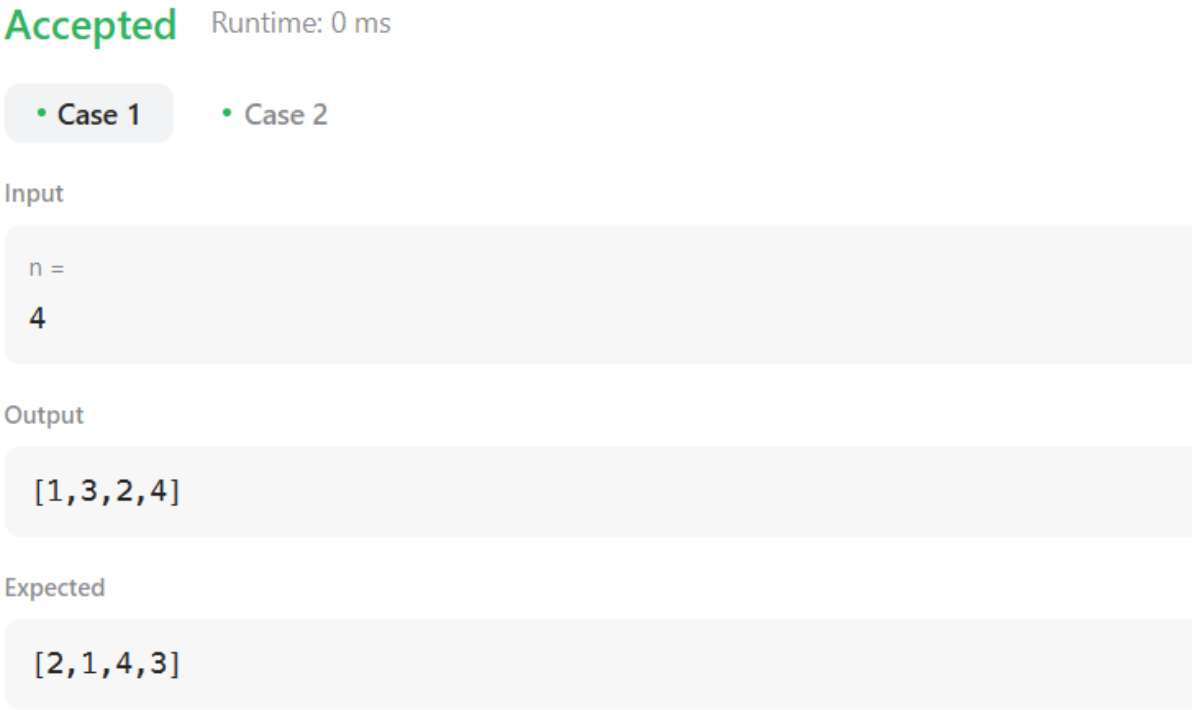

Fig 7. Beautiful Array

5. Learning Outcomes:

- Mastered linked lists and advanced data structures like Fenwick Trees, segment trees, and heaps.
- Improved searching and sorting skills using binary search and merge sort-based counting.
- Learned bit manipulation techniques for efficient operations on binary representations.
- Applied modular arithmetic to handle large computations in problems like Super Pow.
- Solved complex computational geometry problems using priority queues and sweep line algorithms.