Assignment-4

Student Name: Lakhan Singh
UID: 22BCS12194
Branch: BE-CSE
Section/Group: 608-B
Semester: 6th
Date of Performance: 18/03/25
Subject Name: Advance Programming Lab-2
Subject Code: 22CSP-351

1. Aim: Longest Nice Substring.

Code:

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n=s.length();
        if (s.length()<2) {
            return "";
        }
        bool lower[26]={false};
        bool upper[26]={false};
        for(char c:s){
            if(islower(c)){
                lower[c-'a']=true;
            }
            else{
                upper[c-'A']=true;
            }
        }
```

```
        for(int i=0;i<n;i++){
            char c=s[i];
            if(islower(c)&&!upper[c-'a']){
                string left=longestNiceSubstring(s.substr(0,i));
                string right=longestNiceSubstring(s.substr(i+1));
                return left.length()>=right.length()?left:right;
            }
            if(isupper(c)&&!lower[c-'A']){
                string left=longestNiceSubstring(s.substr(0,i));
                string right=longestNiceSubstring(s.substr(i+1));
                return left.length()>=right.length()?left:right;
            }
        }
        return s;
    }
};
```

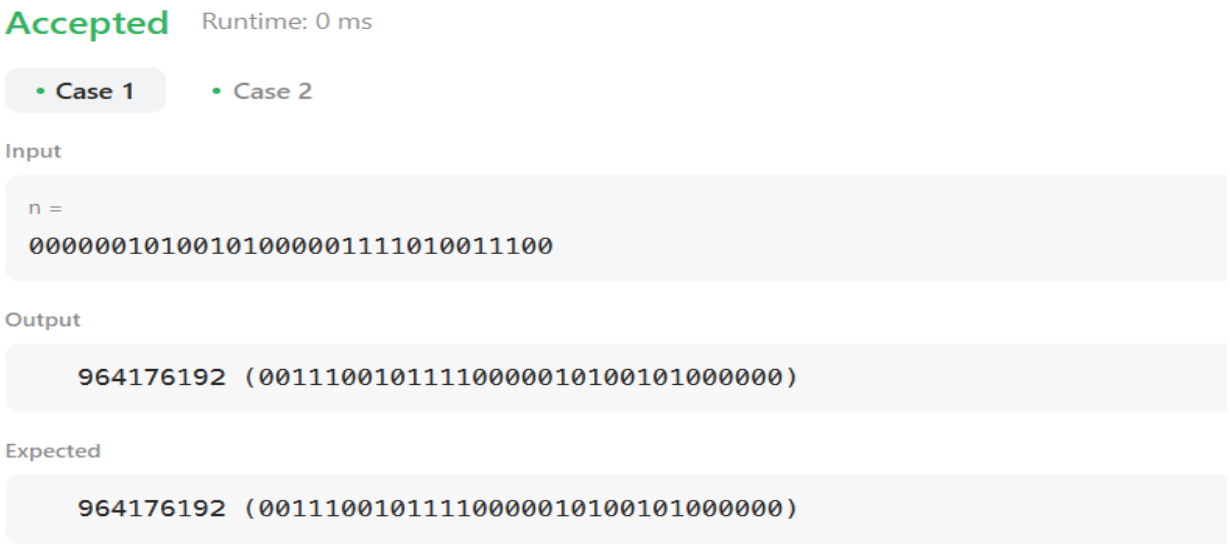
Output:

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++) {
            result = (result << 1) | (n & 1); // Shift result left and add last bit of n
            n >>= 1; // Shift n right to process the next bit
        }
        return result;
    }
};
```

Output:



3. Aim: Number of 1 Bits.

Code:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            n &= (n - 1); // Removes the rightmost set bit
            count++;
        }
        return count;
    }
};
```

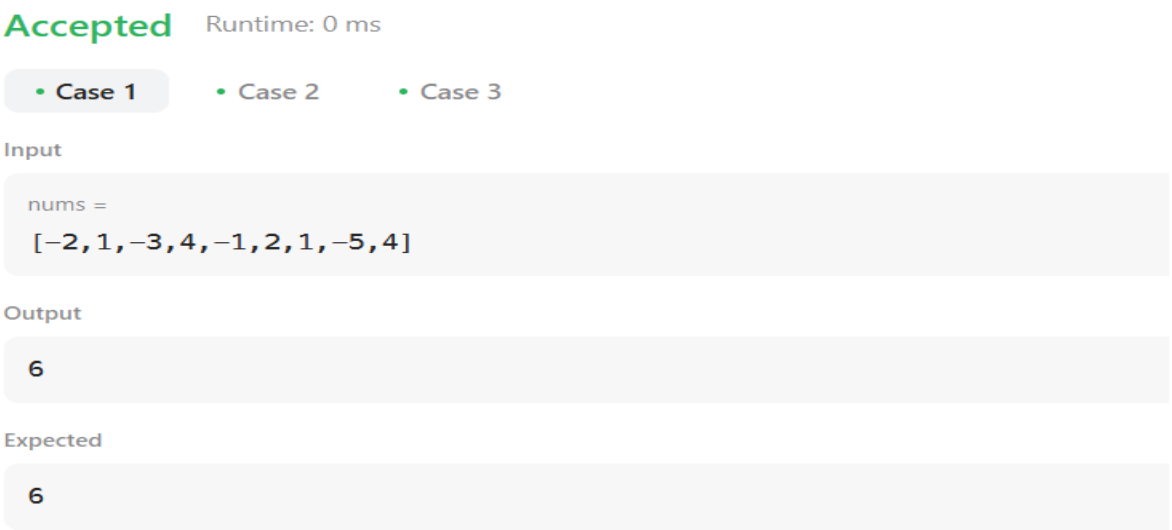
Output:

4. Aim: Maximum Subarray.

Code:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = 0;
        for (int num : nums) {
            currentSum += num;
            maxSum = max(maxSum, currentSum);
            if (currentSum < 0) currentSum = 0;
        }
        return maxSum;
    }
};
```

Output:

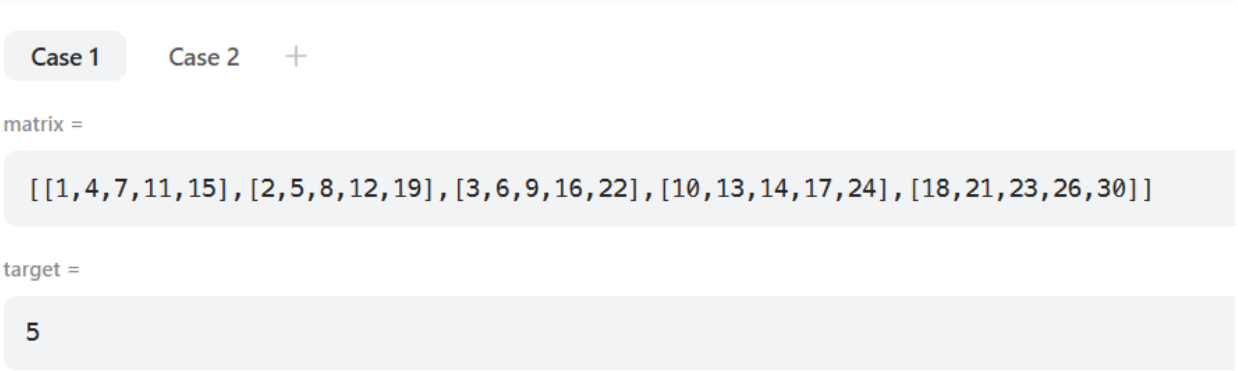


5. Aim: Search a 2D Matrix II.

Code:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = matrix[0].size();
        int row = 0, col = n - 1;  // Start from top-right corner
        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;  // Target not found
    }
};
```

Output:



6. Aim: Super Pow.

Code:

```
class Solution {
public:
    const int MOD = 1337;
    // Function to compute (x^y) % MOD using modular exponentiation
    int powerMod(int x, int y) {
        int result = 1;
        x %= MOD;
        while (y > 0) {
            if (y % 2 == 1) {
                result = (result * x) % MOD;
            }
            x = (x * x) % MOD;
            y /= 2;
        }
        return result;
    }
    // Function to compute a^b % 1337
    int superPow(int a, vector<int>& b) {
        int result = 1;
        for (int digit : b) {
            result = powerMod(result, 10) * powerMod(a, digit) % MOD;
        }
```

```
        return result;
    }
};
```

Output:

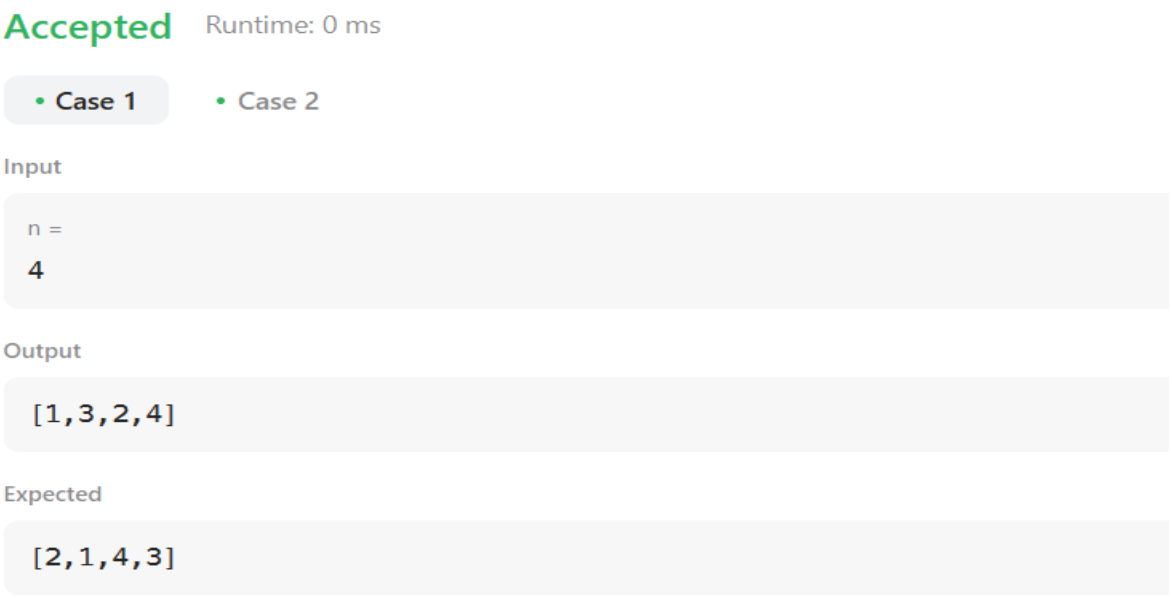
7. Aim: Beautiful Array.

Code:

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> result = {1};  // Base case
        while (result.size() < n) {
            vector<int> temp;

            // Generate odd numbers
```

```
            for (int num : result) {
                if (2 * num - 1 <= n)
                    temp.push_back(2 * num - 1);
            }
            for (int num : result) {
                if (2 * num <= n)
                    temp.push_back(2 * num);
            }
            result = temp;
        }
        return result;
    }
};
```

Output:



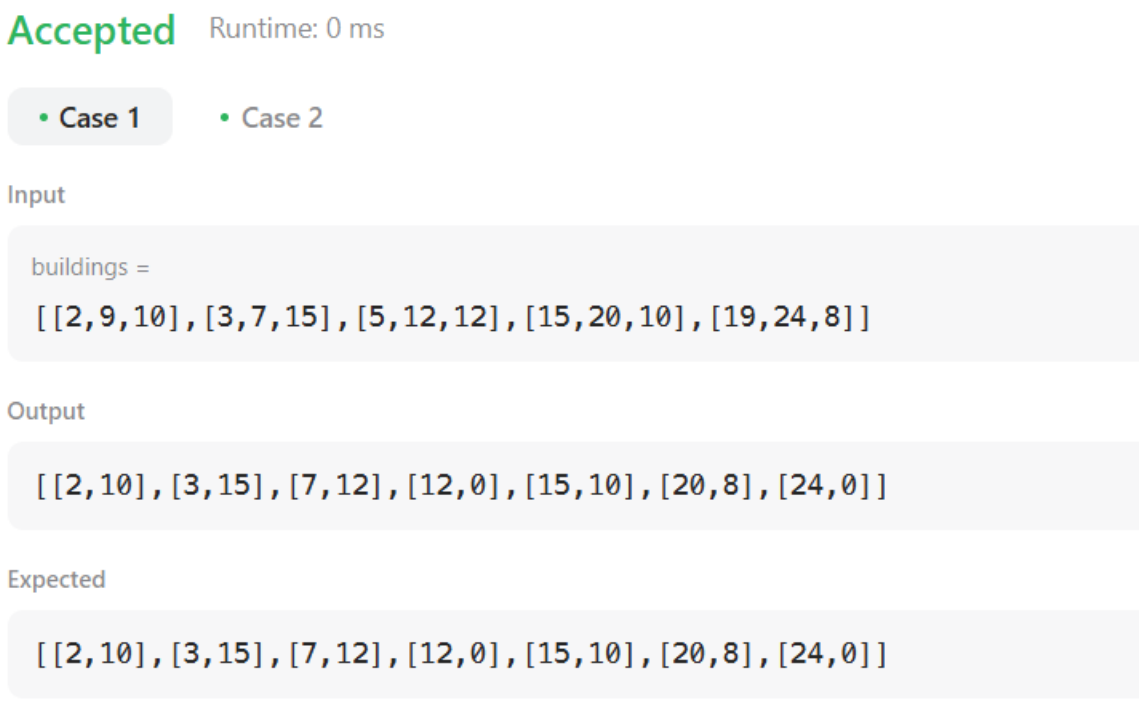
8. Aim: The Skyline Problem.

Code:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;

        for (const auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);
            events.emplace_back(b[1], b[2]);
        }

        sort(events.begin(), events.end());

        vector<vector<int>> result;
        multiset<int> heights = {0};
        int prevMaxHeight = 0;

        for (const auto& event : events) {
            int x = event.first, h = event.second;

            if (h < 0) heights.insert(-h);
            else heights.erase(heights.find(h));
```

```
            int currMaxHeight = *heights.rbegin();

            if (currMaxHeight != prevMaxHeight) {
                result.push_back({x, currMaxHeight});
                prevMaxHeight = currMaxHeight;
            }
        }
        return result;
    }
};
```

Output:



9. Aim: Reverse Pairs.

Code:

```
class Solution {
public:
    int reversePairs(vector<int>& nums) {
        return mergeSort(nums, 0, nums.size() - 1);
    }

private:
    int mergeSort(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2LL * nums[j]) j++;
            count += (j - (mid + 1));
        }
        merge(nums, left, mid, right);
        return count;
    }
    void merge(vector<int>& nums, int left, int mid, int right) {
        vector<int> temp;
        int i = left, j = mid + 1;
```

```
        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) temp.push_back(nums[i++]);
            else temp.push_back(nums[j++]);
        }
        while (i <= mid) temp.push_back(nums[i++]);
        while (j <= right) temp.push_back(nums[j++]);

        for (int k = 0; k < temp.size(); k++) {
            nums[left + k] = temp[k];
        }
    }
};
```

Output:

10. Aim: Longest Increasing Subsequence II.

Code:

```
class Solution {
    vector<int> tree;
    int size;
    int query(int l, int r, int node, int nodeL, int nodeR) {
        if (r < nodeL || nodeR < l) return 0;
        if (l <= nodeL && nodeR <= r) return tree[node];
        int mid = (nodeL + nodeR) / 2;
        return max(query(l, r, 2 * node + 1, nodeL, mid),
                   query(l, r, 2 * node + 2, mid + 1, nodeR));
    }
    void update(int index, int value, int node, int nodeL, int nodeR) {
        if (nodeL == nodeR) {
            tree[node] = value;
            return;
        }
        int mid = (nodeL + nodeR) / 2;
        if (index <= mid) update(index, value, 2 * node + 1, nodeL, mid);
        else update(index, value, 2 * node + 2, mid + 1, nodeR);
        tree[node] = max(tree[2 * node + 1], tree[2 * node + 2]);
    }
public:
    int lengthOfLIS(vector<int>& nums, int k) {
```

```
        int maxNum = 1e5;
        size = maxNum + 1;
        tree.resize(4 * size, 0);
        int maxLength = 0;
        for (int num : nums) {
            int bestPrev = query(max(1, num - k), num - 1, 0, 0, size - 1);
            int newLength = bestPrev + 1;
            update(num, newLength, 0, 0, size - 1);
            maxLength = max(maxLength, newLength);
        }
        return maxLength;
    }
};
```

Output: