EXPERIMENT - 4

Student Name: ANUSHKA

UID:22BET10126

Branch: BE -IT

Section/Group:22BET_IOT-702(B)

Semester: 6th

Subject Code: 22ITP-351

PROBLEM-1

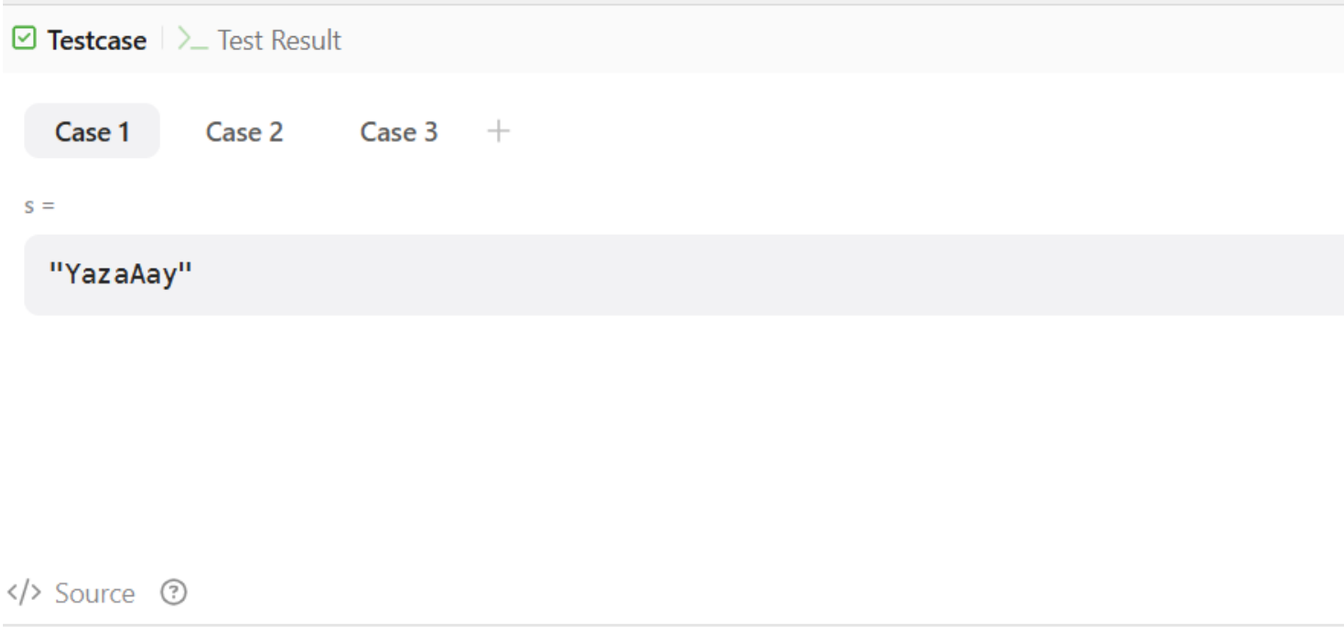
AIM:-

Longest Nice Substring

CODE:-

```
class Solution:
    def longestNiceSubstring(self, s):
        len_s = len(s)
        if len_s <= 1:
            return ''

        char_to_freq_map = defaultdict(int)
        for c in s:
            char_to_freq_map[c] += 1

        is_broken = False
        i = 0
        while (i < len(s)):
            if s[i].islower() and s[i].upper() in char_to_freq_map.keys():
                pass
            elif s[i].isupper() and s[i].lower() in char_to_freq_map.keys():
                pass
            else:
                is_broken = True
                break
            i += 1

        if not is_broken:
            return s

        longest_nice_substr_1 = self.longestNiceSubstring(s[:i])
        longest_nice_substr_2 = self.longestNiceSubstring(s[i+1:])

        if len(longest_nice_substr_1)>=len(longest_nice_substr_2):
            return longest_nice_substr_1
        else:
            return longest_nice_substr_2
```

OUTPUT:-



PROBLEM-2

AIM:-

Reverse bits

CODE:-

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

OUTPUT:-

Testcase | Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

n =

00000010100101000001111010011100

Output

964176192 (00111001011110000010100101000000)

Testcase | Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

n =

11111111111111111111111111111101

Output

3221225471 (10111111111111111111111111111111)

PROBLEM-3

AIM:-

Number of 1 Bits

CODE:-

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

    }
};
```

OUTPUT:-

PROBLEM-4

AIM:-

Maximum Subarray

CODE:-

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

        for (int i = 0; i < nums.size(); i++) {
            currentSum += nums[i];

            if (currentSum > maxSum) {
                maxSum = currentSum;
            }

            if (currentSum < 0) {
```

```
                currentSum = 0;
            }
        }

        return maxSum;
    }
};
```

OUTPUT:-

PROBLEM-5

AIM:-

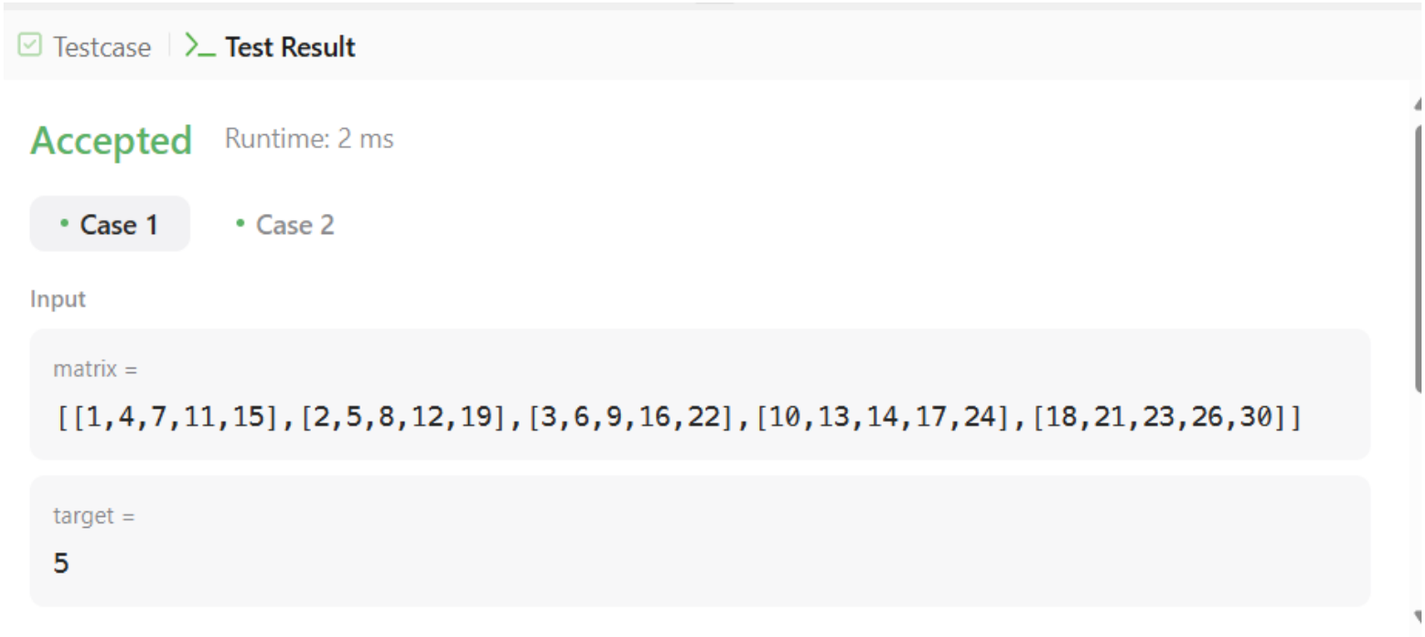
Search a 2D matrix11

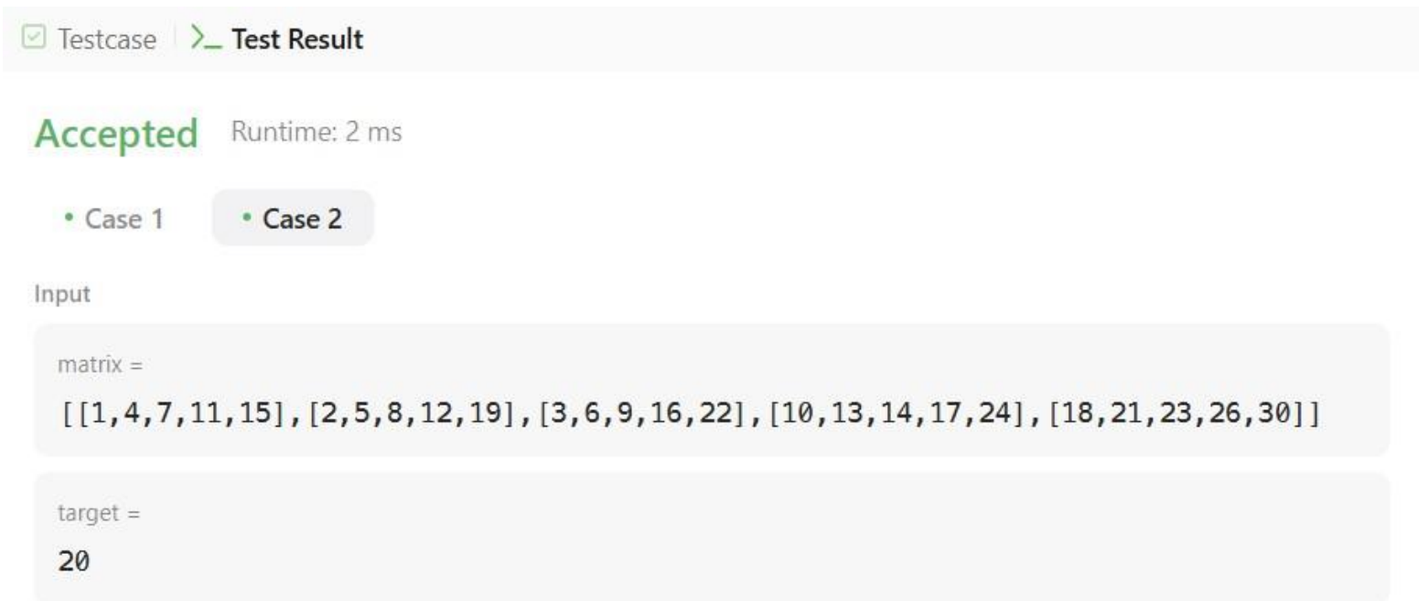
CODE:-

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

```
        int 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:-





PROBLEM-6

AIM:-

Super Pow

CODE:-

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10
    {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)
            result = (result * a) % base;
        return result;
    }
public:
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
    }
};
```

OUTPUT:-

PROBLEM-7

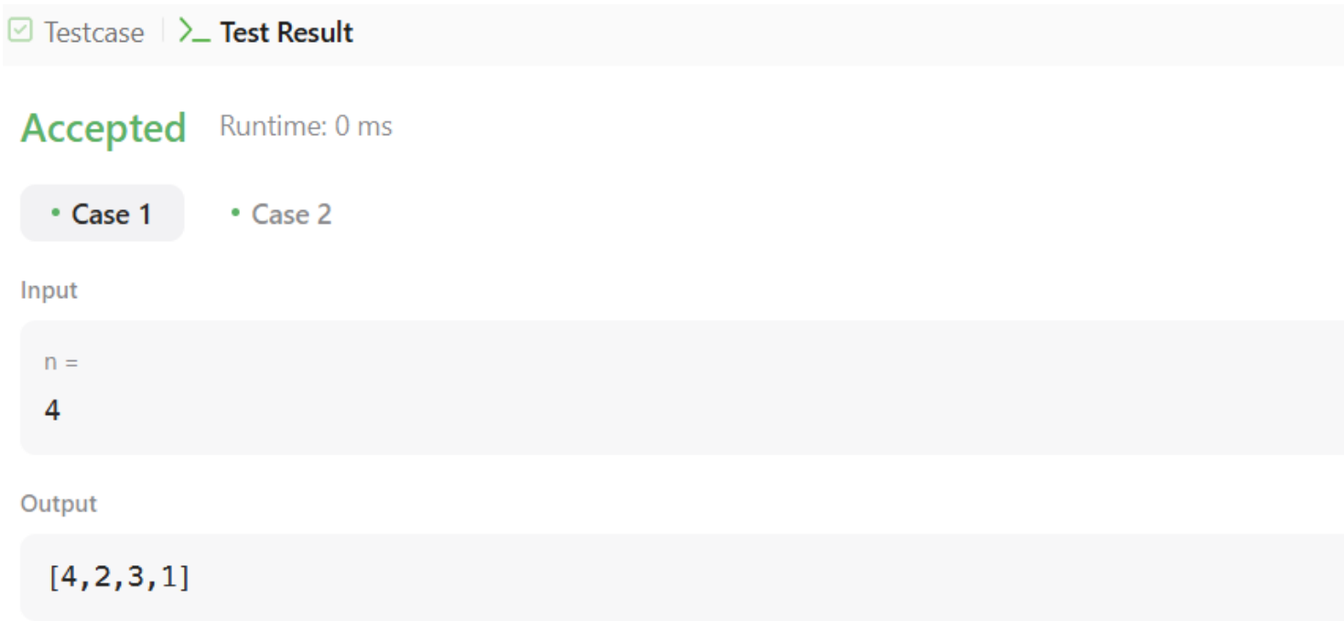
AIM:-

Beautiful Array

CODE:-

```
vector<int> beautifulArray(int N) {
    vector<int> res = {1};
    while (res.size() < N) {
        vector<int> tmp;
        for (int i : res) if (i * 2 - 1 <= N) tmp.push_back(i * 2 - 1);
        for (int i : res) if (i * 2 <= N) tmp.push_back(i * 2);
        res = tmp;
    }
    return res;
}
```

OUTPUT:-



PROBLEM-8

AIM:-

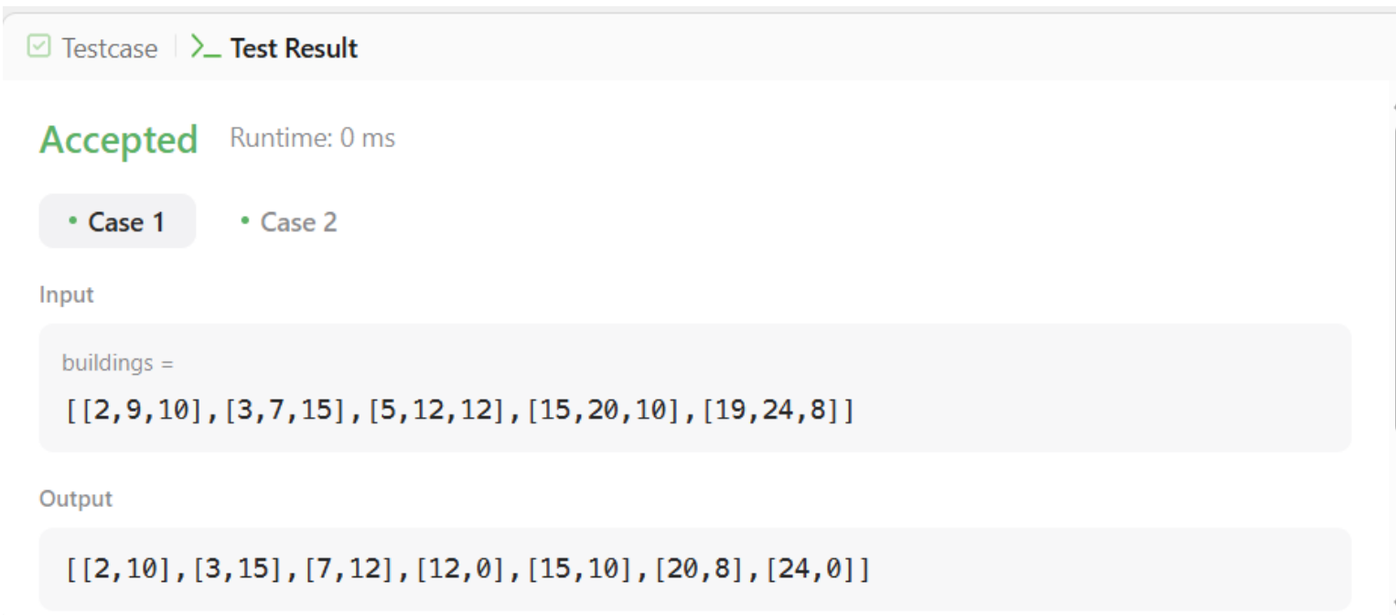
The Skyline Problem

CODE:-

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

        for(auto b: buildings){
            points.push_back({b[0], -b[2]});
            points.push_back({b[1], b[2]});
```

```
        }

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

        int ongoingHeight = 0;

        // points.first = x coordinate, points.second = height
        for(int i = 0; i < points.size(); i++){
            int currentPoint = points[i].first;
            int heightAtCurrentPoint = points[i].second;

            if(heightAtCurrentPoint < 0){
                pq.insert(-heightAtCurrentPoint);
            } else {
                pq.erase(pq.find(heightAtCurrentPoint));
            }

            // after inserting/removing heightAtI, if there's a change
            auto pqTop = *pq.rbegin();
            if(ongoingHeight != pqTop){
                ongoingHeight = pqTop;
                ans.push_back({currentPoint, ongoingHeight});
            }
        }

        return ans;
    }
};
```

OUTPUT:-



PROBLEM-9

AIM:-

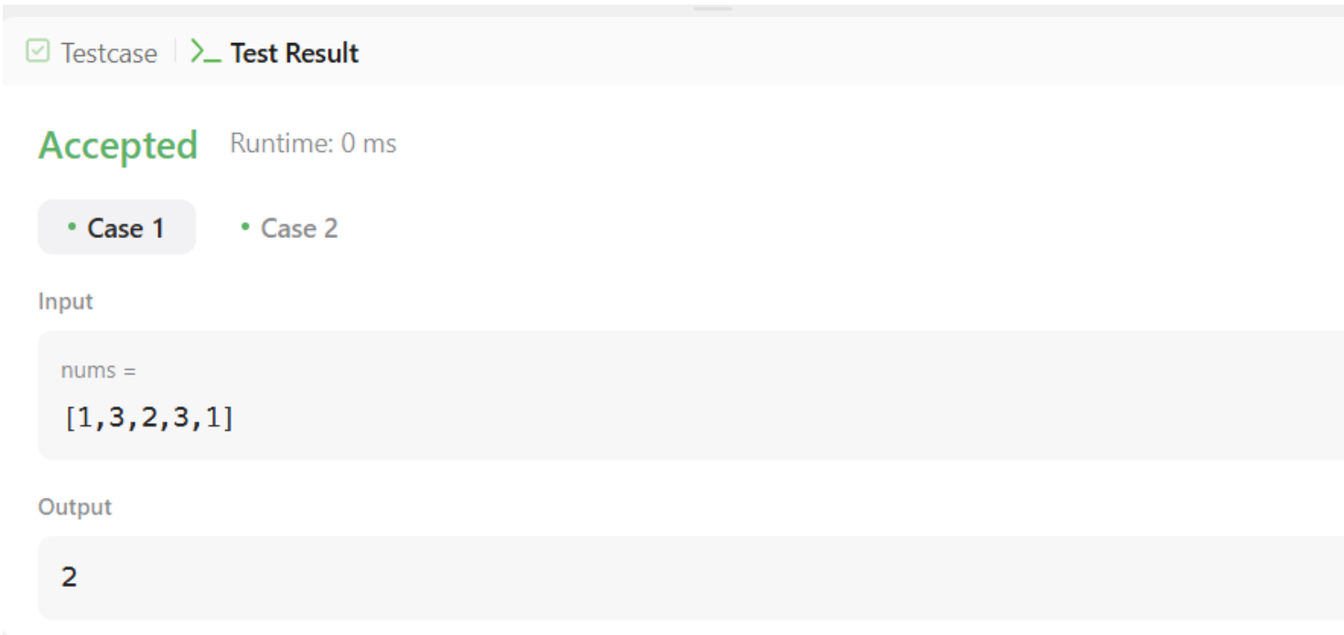
Reverse Pairs

CODE:-

```
class Solution { //OPTIMAL APPROACH,,,T-O(NLOGN * 2), S-O(N)
public:



    void merge(vector<int>&arr, int low, int mid, int high){

        int left = low, right = mid+1; vector<int>temp;
        while(left <= mid && right <= high){
            if(arr[left] <= arr[right]){
                temp.push_back(arr[left++]);
            }
            else{
                temp.push_back(arr[right++]);
            }
        }

        //if left array left
        while(left <= mid){
            temp.push_back(arr[left++]);
        }
        while(right <= high){
            temp.push_back(arr[right++]);
        }

        //now pushing back the temp array elmenets to original array
        for(int i=low; i<=high;i++){
            arr[i] = temp[i - low];
        }

    }



    int countPairs(vector<int>&arr, int low, int mid, int high){
```

```
        int count=0, right=mid+1;
        for(int i = low;i <= mid; i++){
            while (right <= high && (long long)arr[i] > 2LL * arr[right]) right++;
            count += (right - (mid+1));
        }

        return count;
    }

    int mergeSort(vector<int>&arr, int low, int high){
        int count=0;

        if(low >= high) return count;
        int mid = (low + high)/2;
        count += mergeSort(arr, low, mid);
        count += mergeSort(arr, mid+1, high);
        count += countPairs(arr, low, mid, high);
        merge(arr, low, mid, high);

        return count;
    }

    int reversePairs(vector<int>& nums) {
        int n = nums.size();
        return mergeSort(nums, 0, n-1);
    }
};
```

OUTPUT:-



PROBLEM-10

AIM:-

Longest Increasing Subsequesce 11

CODE:-

```
constexpr int N = 100001;
class Solution {
public:
    array<int, 2*N> seg{};

    void update(int pos, int val){ // update max
        pos += N;
        seg[pos] = val;

        while (pos > 1) {
            pos >>= 1;
            seg[pos] = max(seg[2*pos], seg[2*pos+1]);
        }
    }

    int query(int lo, int hi){ // query max [lo, hi)
        lo += N;
        hi += N;
        int res = 0;

        while (lo < hi) {
            if (lo & 1) {
                res = max(res, seg[lo++]);
            }
            if (hi & 1) {
                res = max(res, seg[--hi]);
            }
            lo >>= 1;
            hi >>= 1;
        }
        return res;
    }

    int lengthOfLIS(vector<int>& A, int k) {
        int ans = 0;
```

```
        for (int i = 0; i < size(A); ++i){
            int l = max(0, A[i]-k);
            int r = A[i];
            int res = query(l, r) + 1; // best res for the current element
            ans = max(res, ans);
            update(A[i], res); // and update it here
        }

        return ans;
    }
};
        }
```

OUTPUT:-