**EXPERIMENT4**

**Student Name:** Piyush Prashar **UID:**22BET10096

**Branch:** BE -IT **Section/Group:**22BET\_IOT-703(A)

**Semester:** 6th **Subject Code:** 22ITP-351

**PROBLEM-1**

**AIM:-**

[Longest Nice Substring](https://leetcode.com/problems/longest-nice-substring/)

**CODE:-**

class Solution {

    public String longestNiceSubstring(String s) {

        Set<Character> charSet = new HashSet<>();

        for (int i = 0; i < s.length(); i++) {

            charSet.add(s.charAt(i));

        }

        for (int i = 0; i < s.length(); i++) {

            if (charSet.contains(Character.toUpperCase(s.charAt(i))) &&

                    charSet.contains(Character.toLowerCase(s.charAt(i)))) {

                continue;

            }

            String s1 = longestNiceSubstring(s.substring(0, i));

            String s2 = longestNiceSubstring(s.substring(i+1));

            return s1.length()>= s2.length() ? s1 : s2;

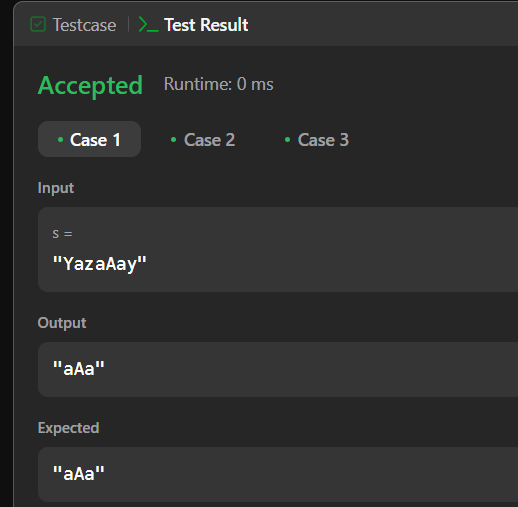
        }

        return s;

    }

}

**OUTPUT:-**

****

**PROBLEM-2**

**AIM:-**

[Reverse Bits](https://leetcode.com/problems/reverse-bits/)

**CODE:**-

public class Solution {

public int reverseBits(int num) {

num = ((num & 0xffff0000) >>> 16) | ((num & 0x0000ffff) << 16);

num = ((num & 0xff00ff00) >>> 8) | ((num & 0x00ff00ff) << 8);

num = ((num & 0xf0f0f0f0) >>> 4) | ((num & 0x0f0f0f0f) << 4);

num = ((num & 0xcccccccc) >>> 2) | ((num & 0x33333333) << 2);

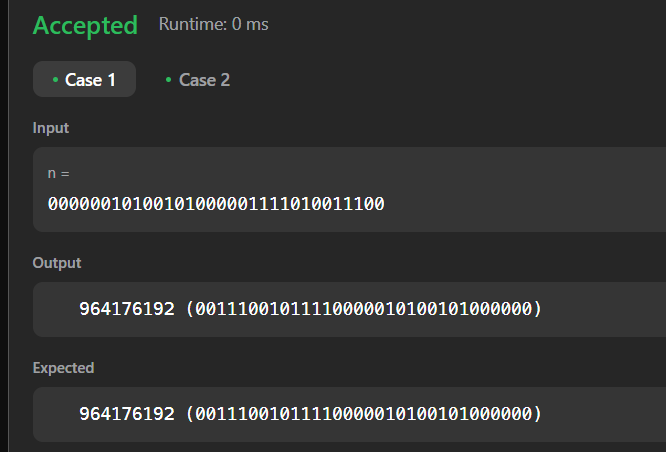
num = ((num & 0xaaaaaaaa) >>> 1) | ((num & 0x55555555) << 1);

return num;

}

}

**OUTPUT:-**

****

**PROBLEM-3**

**AIM:-**

[Number of 1 Bits](https://leetcode.com/problems/number-of-1-bits/)

**CODE:-**

class Solution {

public int hammingWeight(int n) {

int set = 0;

while(n > 0){

n = (n & (n-1));

set++;

}

return set;

}

}

**OUTPUT:-**

****

**PROBLEM-4**

**AIM:-**

Maximum Subarray

**CODE:-**

class Solution {

    public int maxSubArray(int[] nums) {

        int res = nums[0];

        int total = 0;

        for (int n : nums) {

            if (total < 0) {

                total = 0;

            }

            total += n;

            res = Math.max(res, total);

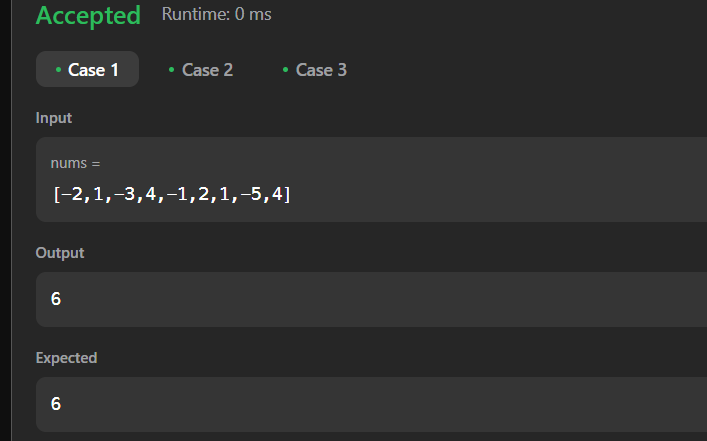
        }

        return res;

    }

}

**OUTPUT:-**

****

**PROBLEM-5**

**AIM:-**

Search 2d matrix 2

**CODE:-**

public class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

        if(matrix == null || matrix.length < 1 || matrix[0].length <1) {

            return false;

        }

        int col = matrix[0].length-1;

        int row = 0;

        while(col >= 0 && row <= matrix.length-1) {

            if(target == matrix[row][col]) {

                return true;

            } else if(target < matrix[row][col]) {

                col--;

            } else if(target > matrix[row][col]) {

                row++;

            }

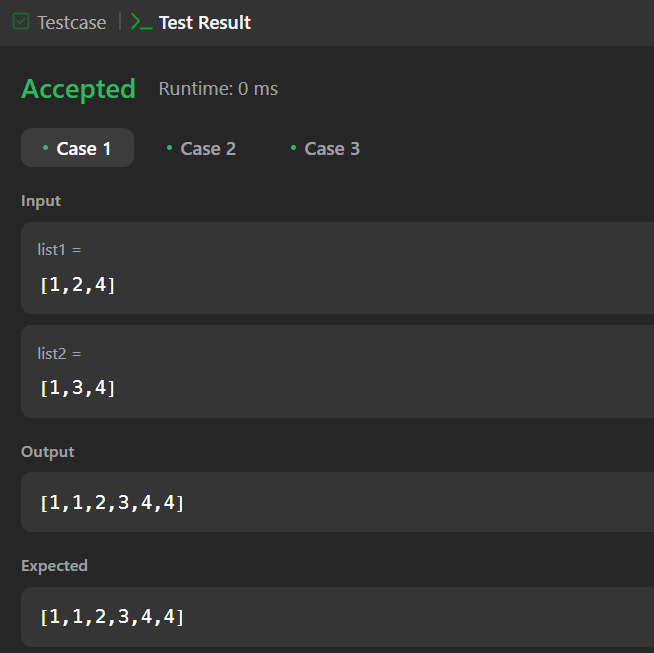
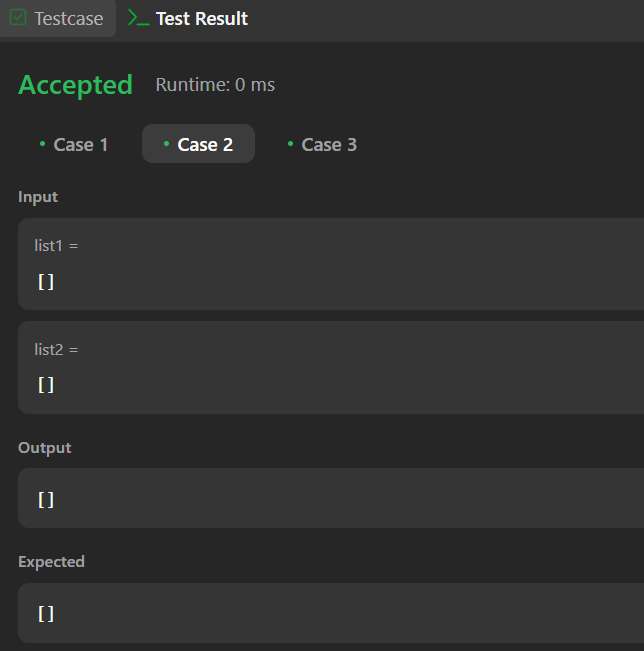
        }

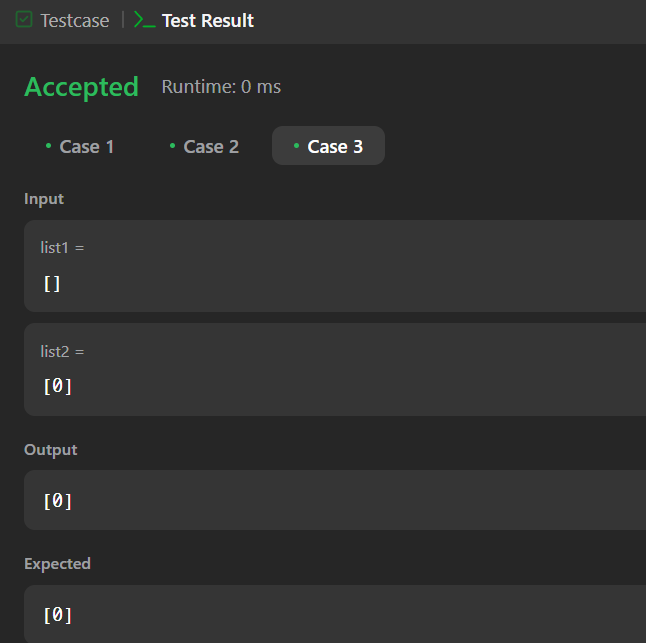
        return false;

    }

}

**OUTPUT:-**

** **

****

**PROBLEM-6**

**AIM:-**

Super Pow

**CODE:-**

class Solution {

public int superPow(int a, int[] b) {

int num=0;

for(int i:b){

num=(num\*10+i)%1140;

}

return binexpo(a,num,1337);

}

public int binexpo(int a, int b, int m){

a%=m;

int res=1;

while(b>0){

if((b&1)==1)

res=(res\*a)%m;

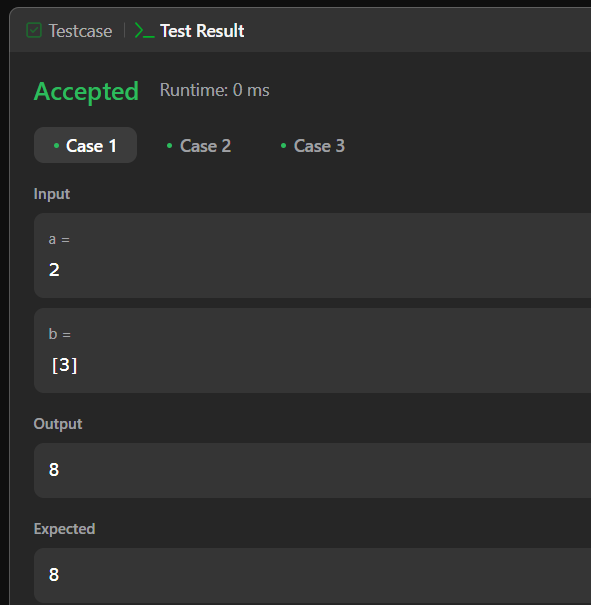
a=(a\*a)%m;

b>>=1;

}

return res;

}

} **OUTPUT:- **

**PROBLEM-7**

**AIM:-**

Beautiful Array

**CODE:-**

class Solution {

public int[] beautifulArray(int n) {

int[] ans = new int[n];

for(int i = 0; i < n; i++){

ans[i] = i+1;

}

recursion(ans, 0, n-1);

return ans;

}

public void recursion(int[] arr, int left, int right){

if(left >= right)

return;

ArrayList<Integer> l = new ArrayList<>();

ArrayList<Integer> r = new ArrayList<>();

boolean alt = true;// Not worry about whether the factor of the interval is even or odd too much, they can be grouped by

// just picking one and skip one

for(int i = left; i <= right; i++){ // picking the elements and put them into the two groups

if(alt)

l.add(arr[i]);

else

r.add(arr[i]);

alt = !alt;

}

for(int i = left; i <= right; i++){ // merging them into the final array

if(!l.isEmpty())

arr[i] = l.remove(0);

else

arr[i] = r.remove(0);

}

recursion(arr, left, (right+left)/2);

recursion(arr, (left+right)/2+1, right);

}

} OUTPUT:-

****

**PROBLEM-8**

**AIM:-**

The Skyline Problem

**CODE:-**

class TopNode {

int x;

int h;

TopNode next;

TopNode() {

}

TopNode(int x, int h) {

this.x = x;

this.h = h;

}

void insert(TopNode n) {

n.next = next;

next = n;

}

}

class Solution {

static final int LEFT=0, RIGHT=1, HEIGHT=2;

public List<List<Integer>> getSkyline(int[][] buildings) {

TopNode head = new TopNode(0,0);

head.insert(new TopNode(Integer.MAX\_VALUE, 0));

TopNode start = head;

for (int i = 0; i<buildings.length; i++) {

int[] b = buildings[i];

int bL = buildings[i][LEFT];

int bR = buildings[i][RIGHT];

int bH = buildings[i][HEIGHT];

//System.out.println(Arrays.toString(buildings[i]));

while (bL >= start.next.x) { start = start.next; }

//System.out.println(start.toString());

for (TopNode t = start ; bR > t.x; t = t.next) {

//System.out.println(head.toString());

if (bH <= t.h) {

continue;

}

TopNode stop = t;

while (stop.next != null && stop.next.x < bR && stop.next.h <= bH ) {

stop = stop.next;

}

if (bL <= t.x) {

if (bR >= stop.next.x) {

t.next = stop.next;

t.h = bH;

}

else if (t == stop) {

t.insert(new TopNode(bR,t.h));

t.h = bH;

break;

}

else {

stop.x = bR;

t.h = bH;

t.next = stop;

break;

}

}

else {

if (bR >= stop.next.x) {

if (t == stop) {

t.insert(new TopNode(bL, bH));

}

else {

t.next = stop;

stop.x = bL;

stop.h = bH;

}

break;

}

else if (t == stop) {

t.insert(new TopNode(bL,bH));

t.next.insert(new TopNode(bR,t.h));

break;

}

else {

t.next = stop;

t.insert(new TopNode(bL,bH));

stop.x = bR;

break;

}

}

t = stop;

}

}

List<List<Integer>> skyline = new ArrayList<>();

if (head.h == 0)

head = head.next;

while (head != null) {

int height = head.h;

skyline.add(List.of(head.x, height));

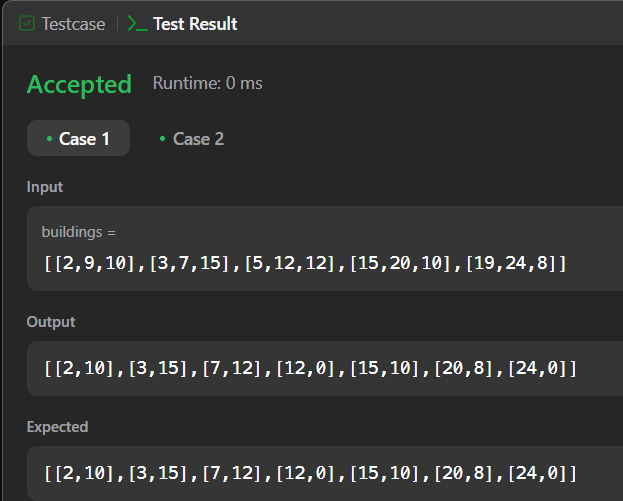
while ( (head = head.next) != null && head.h == height) {}

}

return skyline;

}

} **OUTPUT:-**

****

**PROBLEM-9**

**AIM:-**

Reverse Pairs

**CODE:-**

class Solution {

public void merge(int[] arr, int low, int mid, int high) {

ArrayList<Integer> temp = new ArrayList<>();

int left = low;

int right = mid+1;

while(left <= mid && right <= high) {

if(arr[left] <= arr[right]) {

temp.add(arr[left++]);

} else {

temp.add(arr[right++]);

}

}

while(left <= mid) temp.add(arr[left++]);

while(right <= high) temp.add(arr[right++]);

for(int i=low; i<=high; i++) {

arr[i] = temp.get(i-low);

}

}

public int countPairs(int[] arr, int low, int mid, int high) {

int right = mid + 1;

int cnt = 0;

for(int i=low; i<=mid; i++) {

while(right <= high && (long) arr[i] > 2L \* arr[right])

right++;

cnt += (right - (mid + 1));

}

return cnt;

}

public int mergeSort(int[] arr, int low, int high) {

int cnt = 0;

if(low >= high) return cnt;

int mid = (low + high) / 2;

cnt += mergeSort(arr,low,mid);

cnt += mergeSort(arr,mid+1,high);

cnt += countPairs(arr,low,mid,high);

merge(arr,low,mid,high);

return cnt;

}

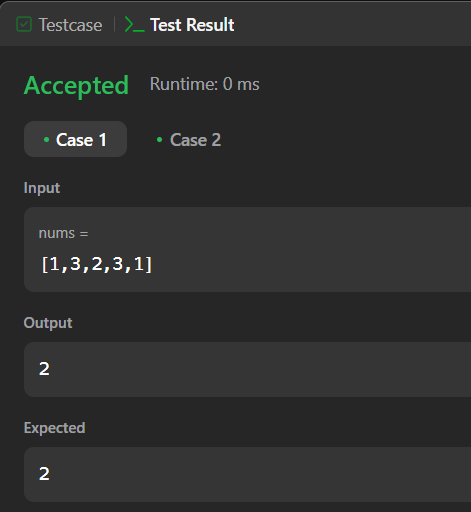
public int reversePairs(int[] nums) {

int n = nums.length;

return mergeSort(nums, 0, n-1);

}

} **OUTPUT:-**

****

**PROBLEM-10**

**AIM:-**

Longest increasing subsequence 2

**CODE:-**

class Solution {

public int lengthOfLIS(int[] nums, int k) {

int[] temp = new int[nums.length];

int ans = 1;

Arrays.fill(temp, 1);

for(int i = 1; i < nums.length; i++) {

for(int j = 0; j < i; j++) {

if(nums[i] > nums[j] && nums[i] - nums[j] <= k) {

temp[i] = Math.max(temp[i], temp[j] + 1);

ans = Math.max(temp[i], ans);

}

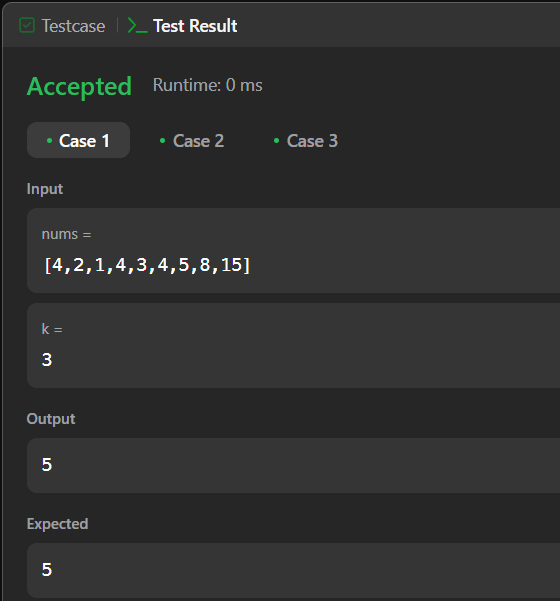
}

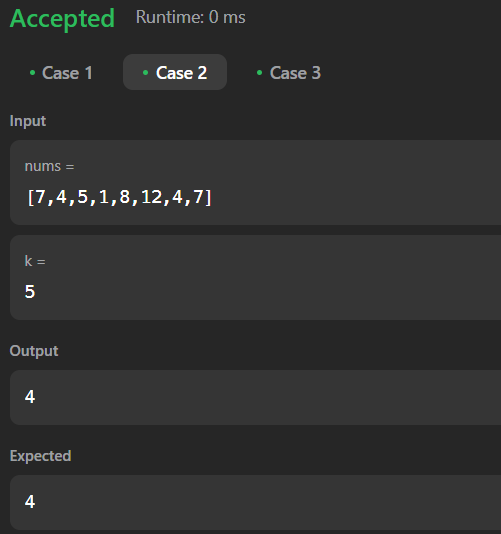
}

return ans;

}

}

**OUTPUT:- **

****