Name: Devesh

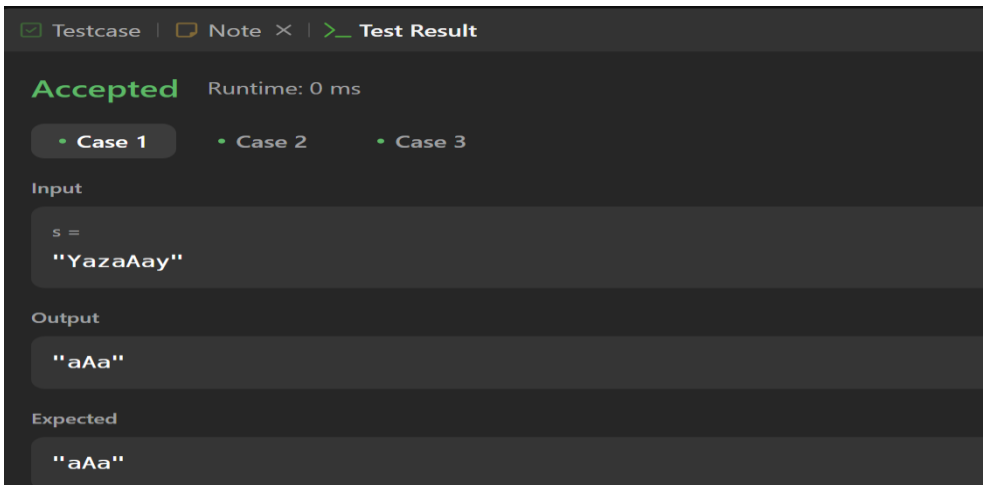
UID: 22BCS16690

Class : 605 - B

Q 1 Longest Nice Substring

```
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:



Q 2 Reverse Bits

```
public class Solution {
    public int reverseBits(int n) {
        int result = 0;
        for(int i = 0; i<32; i++){
            int lsb = n & 1;
            int reverse = lsb <<(31-i);
             result = result | reverse;
             n = n >> 1;
        }
        return result;
    }
}
```

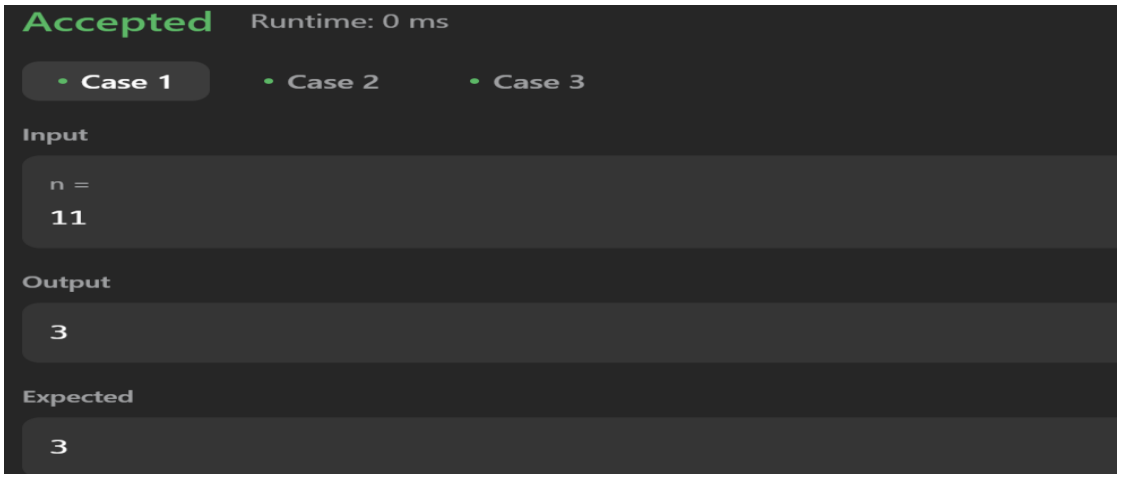
OUTPUT:

Q3 Number of 1 Bits

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while(n>0){
            if((n & 1) != 0){
                count++;
            }
```

```
            n = n>>1;
        }
        return count;
    }
}
```

OUTPUT:

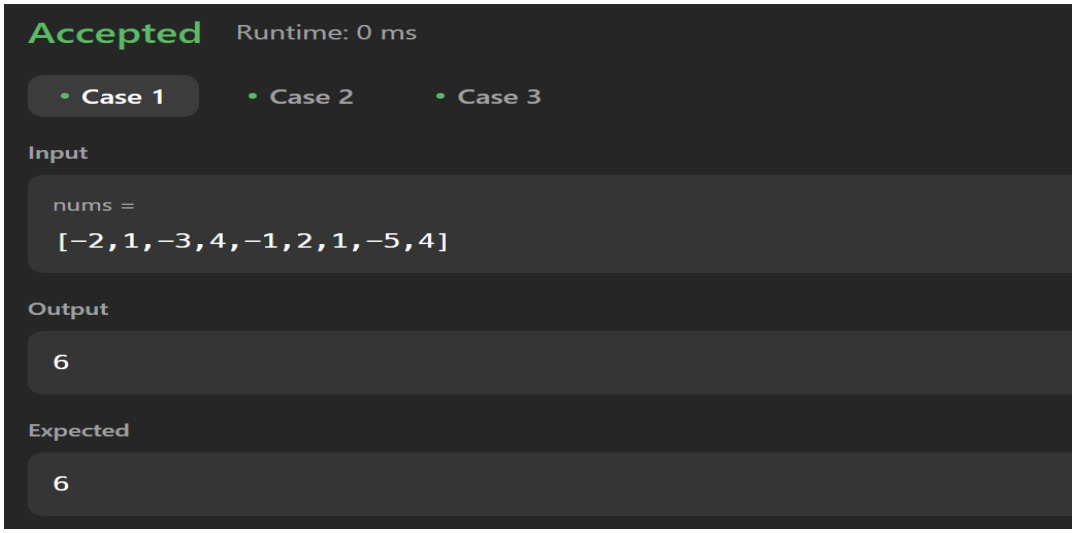


Q 4 Maximum Subarray

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = Integer.MIN_VALUE;
        int currentSum = 0;

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

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

            if (currentSum < 0) {
                currentSum = 0;
            }
        }
```

```
        return maxSum;
    }
}
```

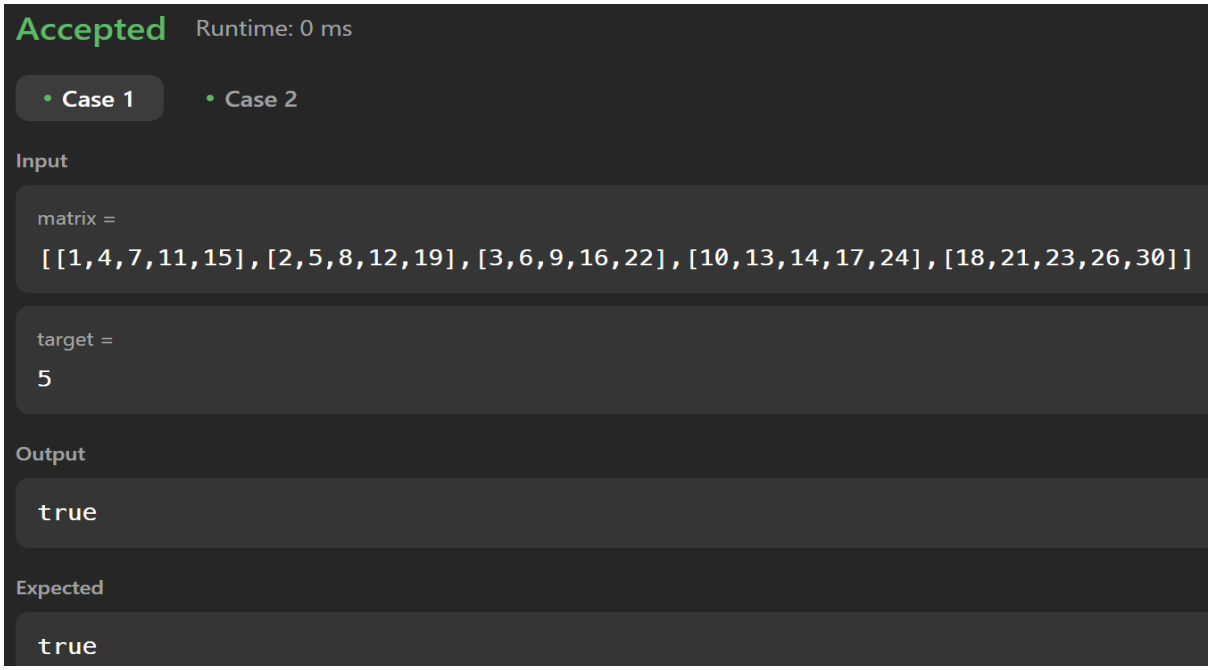
OUTPUT:



Q 5 Search a 2D Matrix II

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int r = rows - 1;
        int c = 0;
        while (r >= 0 && c < cols) {
            if (matrix[r][c] > target) {
                r --;
            } else if (matrix[r][c] < target) {
                c ++;
            } else {
                return true;
            }
        }
        return false;
    }
}
```

OUTPUT:



Q 6 Super Pow

```
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:

Q 7 Beautiful Array

```
class Solution {
    public int[] beautifulArray(int N) {
        int[] res = new int[N];
        if (N == 1)
        {
            return new int[] {1};
        }
        else if (N == 2)
        {
            return new int[] {1, 2};
        }
        else
        {
            int[] odds = beautifulArray((N + 1) / 2);
            int[] even = beautifulArray(N / 2);
            for (int i = 0; i < odds.length; i ++)
            {
```

```
            res[i] = odds[i] * 2 - 1;
        }
        for (int j = 0; j < even.length; j ++)
        {
            res[odds.length + j] = even[j] * 2;
        }
    }
    return res;
  }
}
```

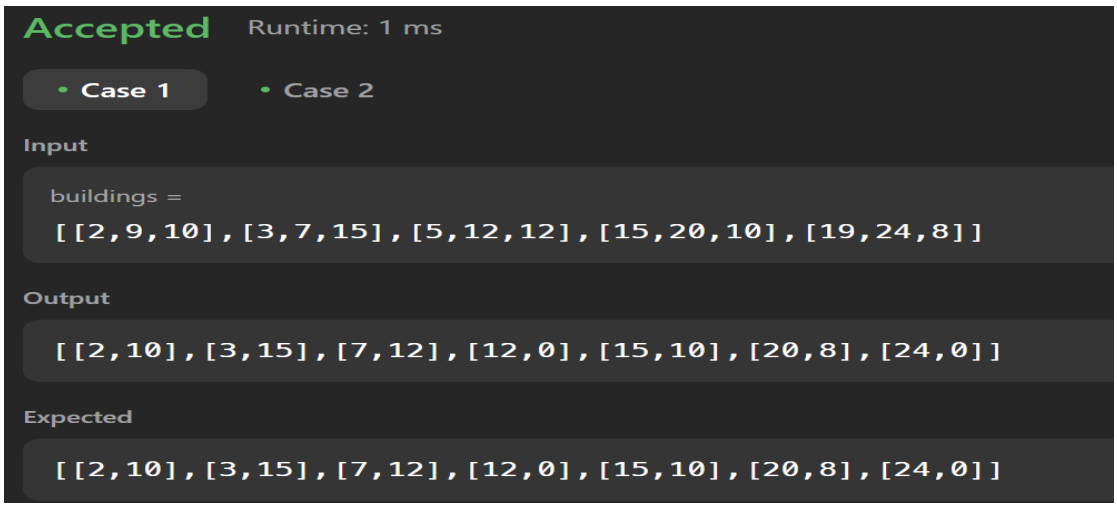
OUTPUT:

Q 8 The Skyline Problem

```
class Solution {
public List<List<Integer>> getSkyline(int[][] buildings) {
    List<List<Integer>> list = new ArrayList<>();

    List<int[]> lines = new ArrayList<>();
    for (int[] building: buildings) {
        lines.add(new int[] {building[0], building[2]});
        lines.add(new int[] {building[1], -building[2]});
    }
    Collections.sort(lines, (a, b)->a[0]==b[0]?b[1]-a[1]:a[0]-b[0]);
```

```
        TreeMap<Integer, Integer> map = new TreeMap<>();
        map.put(0, 1);
        int prev=0;
        for (int[] line: lines) {
            if (line[1]>0) {
                map.put(line[1], map.getOrDefault(line[1], 0)+1);
            } else {
                int f = map.get(-line[1]);
                if (f==1) map.remove(-line[1]);
                else map.put(-line[1], f-1);
            }
            int curr = map.lastKey();
            if (curr!=prev) {
                list.add(Arrays.asList(line[0], curr));
                prev=curr;
            }
        }
        return list;
    }
}
```

OUTPUT:



Q 9 Reverse Pairs

```
class Solution {
    public int reversePairs(int[] nums) {
        int ans = 0;
        List<Long> res = new ArrayList<>();
        res.add((long) nums[nums.length - 1] * 2);

        for (int i = nums.length - 2; i >= 0; i--) {
            ans += LessThanx(res, nums[i]);
            update(res, (long) nums[i] * 2);
        }

        return ans;
    }

    private int LessThanx(List<Long> res, long val) {
        if (res.get(0) >= val) {
            return 0;
        }

        if (res.get(res.size() - 1) < val) {
            return res.size();
        }

        int lo = 0, hi = res.size() - 1;

        while (lo < hi) {
            int mid = (lo + hi) / 2;
```

```
            if (res.get(mid) < val) {
                lo = mid + 1;
            } else {
                hi = mid;
            }
        }

        return lo;
    }

    private void update(List<Long> res, long val) {
        int index = Collections.binarySearch(res, val);
        if (index < 0) {
            index = -(index + 1);
        }
        res.add(index, val);
    }
}
```

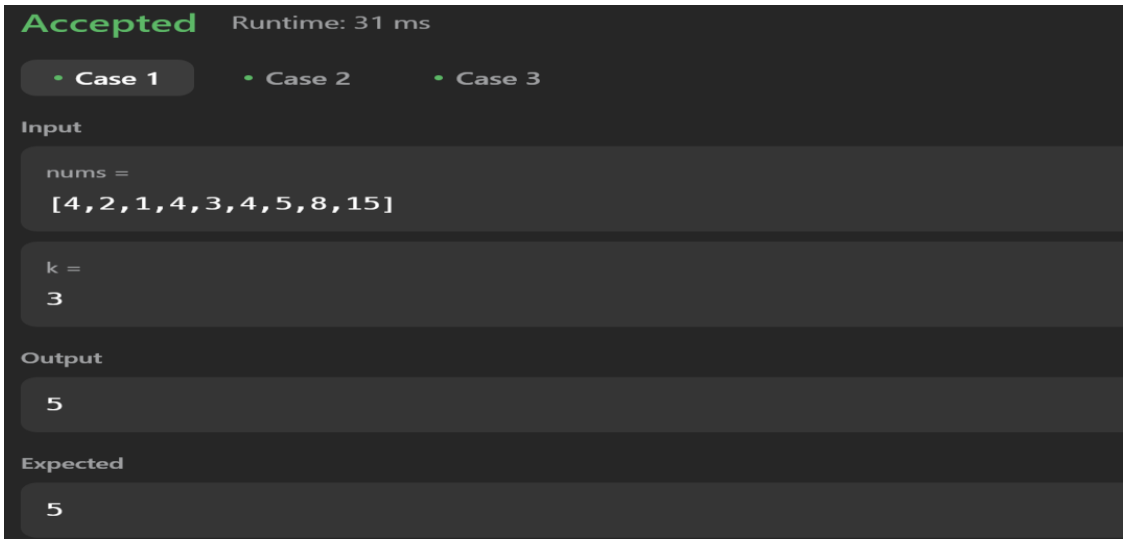
OUTPUT:

Q 10 Longest Increasing Subsequence II

```
class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        SegmentTree root = new SegmentTree(1, 100000);
        int res = 0;
        for (int num : nums) {
            int preMax = root.rangeMaxQuery(root, num - k, num - 1);
            root.update(root, num, preMax + 1);
            res = Math.max(res, preMax + 1);
        }
        return res;
    }
}
class SegmentTree {
    SegmentTree left, right;
    int start, end, val;
    public SegmentTree(int start, int end) {
        this.start = start;
        this.end = end;
        setup(this, start, end);
    }
    public void setup(SegmentTree node, int start, int end) {
        if (start == end) return;
        int mid = start + (end - start) / 2;
        if (node.left == null) {
            node.left = new SegmentTree(start, mid);
            node.right = new SegmentTree(mid + 1, end);
        }
        setup(node.left, start, mid);
```

```
        setup(node.right, mid + 1, end);
        node.val = Math.max(node.left.val, node.right.val);
    }
    public void update(SegmentTree node, int index, int val) {
        if (index < node.start || index > node.end) return;
        if (node.start == node.end && node.start == index) {
            node.val = val;
            return;
        }
        update(node.left, index, val);
        update(node.right, index, val);
        node.val = Math.max(node.left.val, node.right.val);
    }
    public int rangeMaxQuery(SegmentTree node, int start, int end) {
        if (node.start > end || node.end < start) return 0;
        if (node.start >= start && node.end <= end) return node.val;
        return Math.max(rangeMaxQuery(node.left, start, end), rangeMaxQuery(node.right,
start, end));
    }
}
```

OUTPUT:



Q 11 Merge Sorted Array

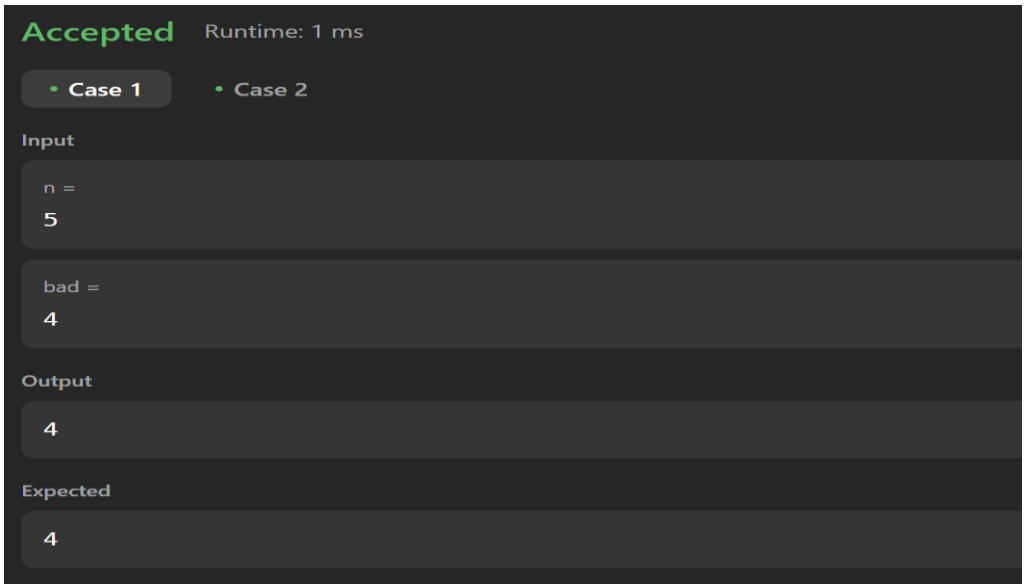
```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int midx = m - 1;
        int nidx = n - 1;
        int right = m + n - 1;

        while (nidx >= 0) {
            if (midx >= 0 && nums1[midx] > nums2[nidx]) {
                nums1[right] = nums1[midx];
                midx--;
            } else {
                nums1[right] = nums2[nidx];
                nidx--;
            }
            right--;
        }       } }
```

OUTPUT:

Q 12 First Bad Version

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int first = 1;
        int last = n;

    while (first <= last) {
      int mid = first + (last - first) / 2;

        if (isBadVersion(mid)) {
            last = mid - 1;
        } else {
            first = mid + 1;
        }
    }
    return first;

    }
}
```

OUTPUT:



Q 13 Sort Colors

```
class Solution {
    public void sortColors(int[] nums) {
        int zeros = 0, ones = 0, n = nums.length;
        for(int num : nums) {
            if(num == 0) zeros++;
            else if(num == 1) ones++;
        }

        for(int i = 0; i < zeros; ++i) {
            nums[i] = 0;
        }

        for(int i = zeros; i < zeros + ones; ++i) {
            nums[i] = 1;
        }

        for(int i = zeros + ones; i < n; ++i) {
            nums[i] = 2;
        } } }
```

OUTPUT:

Q 14 Top K Frequent Elements

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        List<Integer>[] bucket = new List[nums.length + 1];
        HashMap<Integer, Integer> hm = new HashMap<>();
        for (int num : nums) {
            hm.put(num, hm.getOrDefault(num,0) + 1);
        }
        for (int key : hm.keySet()) {
            int freq = hm.get(key);
            if (bucket[freq] == null) {
                bucket[freq] = new ArrayList<>();
            }
            bucket[freq].add(key);
        }
        int[] ans = new int[k];
        int pos = 0;
        for (int i = bucket.length - 1; i >= 0; i--) {
            if (bucket[i] != null) {
                for (int j = 0; j < bucket[i].size() && pos < k; j++) {
                    ans[pos] = bucket[i].get(j);
                    pos++;
                }
            }
        }
        return ans;
    }
}
```

OUTPUT:

Q 15 Kth Largest Element in an Array

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        Arrays.sort(nums);
        return nums[nums.length - k];
    }
}
```

OUTPUT: