1. Longest Nice Substring

class Solution {

String S;

String divide(int l, int r) {

if (l > r) return "";

int[] lowerCase = new int[26];

int[] upperCase = new int[26];

for (int i = l; i <= r; i++) {

char ch = S.charAt(i);

if (Character.isUpperCase(ch)) {

upperCase[ch - 'A']++;

} else {

lowerCase[ch - 'a']++;

}

}

for (int i = l; i <= r; i++) {

char ch = S.charAt(i);

if ((Character.isUpperCase(ch) && lowerCase[ch - 'A'] == 0) ||

(Character.isLowerCase(ch) && upperCase[ch - 'a'] == 0)) {

String leftPart = divide(l, i - 1);

String rightPart = divide(i + 1, r);

return leftPart.length() >= rightPart.length() ? leftPart : rightPart;

}

}

return S.substring(l, r + 1);

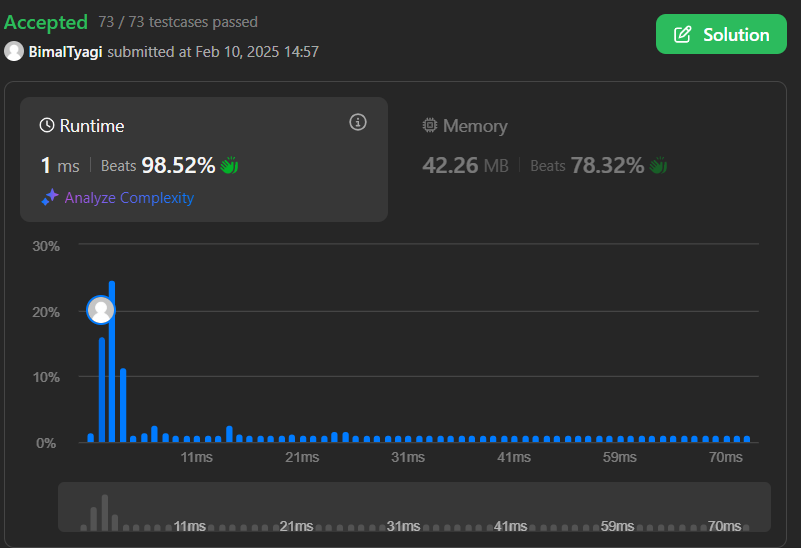
}

public String longestNiceSubstring(String s) {

S = s;

return divide(0, S.length() - 1);

}

}

1. Reverse Bits

public class Solution {

public int reverseBits(int n) {

int result = 0;

for (int i = 0; i < 32; i++) {

int bit = n & 1;

result = (result << 1) | bit;

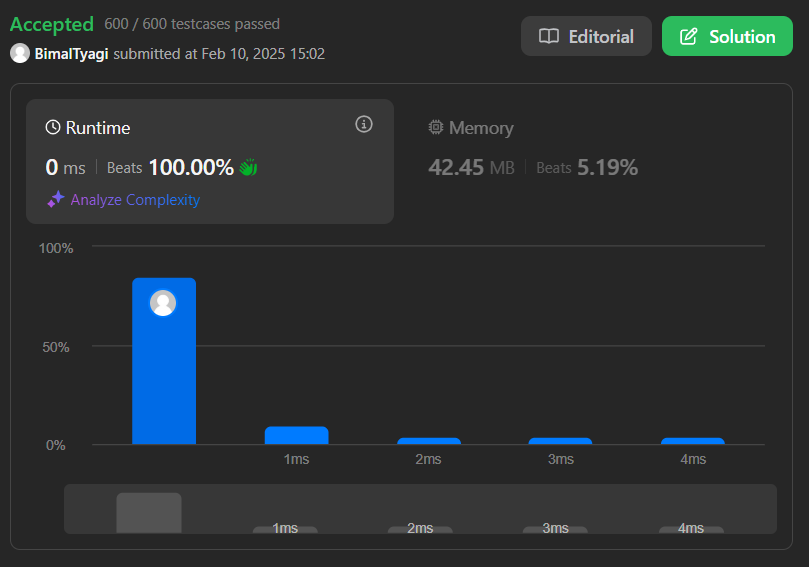
n = n >>> 1;

}

return result;

}

}



1. Number of 1 Bits

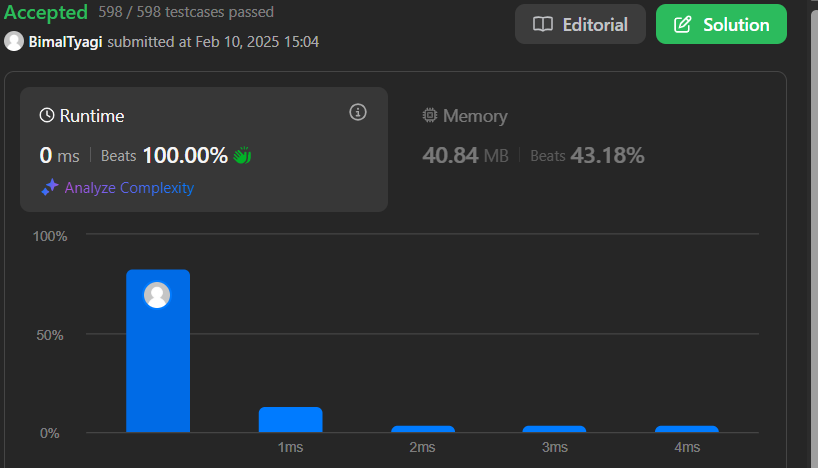
public class Solution {

public int hammingWeight(int n) {

int res = 0;

for (int i = 0; i < 32; i++) {

if (((n >> i) & 1) == 1) {

res += 1;

}

}

return res;

}

}

1. Maximum Subarray

class Solution {

public int maxSubArray(int[] nums) {

int res = nums[0];

int total = 0;

for (int n : nums) {

if (total < 0) {

total = 0;

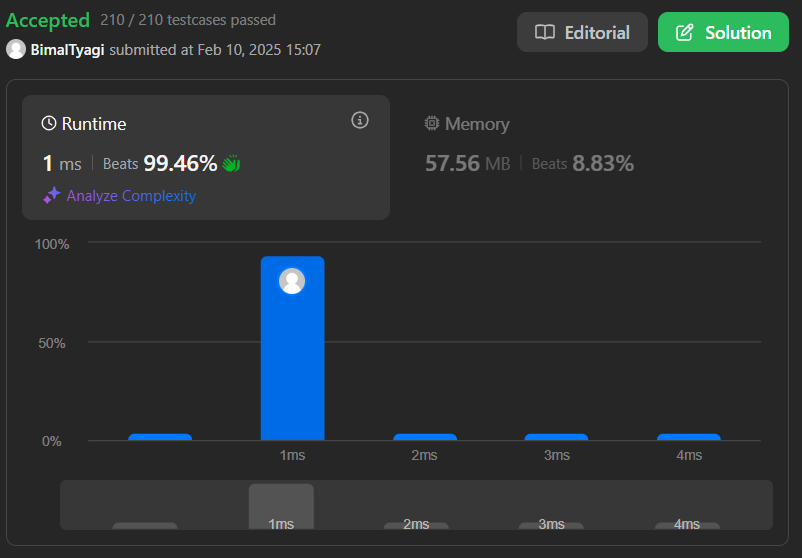
}

total += n;

res = Math.max(res, total);

}

return res;

}

}

1. Search a 2D Matrix II

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++;

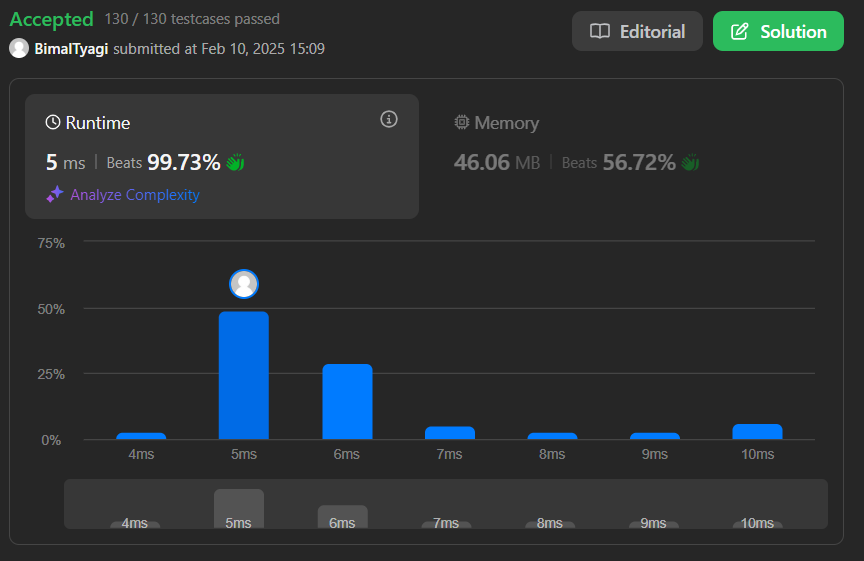
}

}

return false;

}

}



1. Super Pow

class Solution {

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

if (a % 1337 == 0) return 0;

int p = 0;

for (int i : b) p = (p \* 10 + i) % 1140;

if (p == 0) p += 1440;

return power(a, p, 1337);

}

public int power(int a, int n, int mod) {

a %= mod;

int ret = 1;

while (n != 0) {

if ((n & 1) != 0) ret = ret \* a % mod;

a = a \* a % mod;

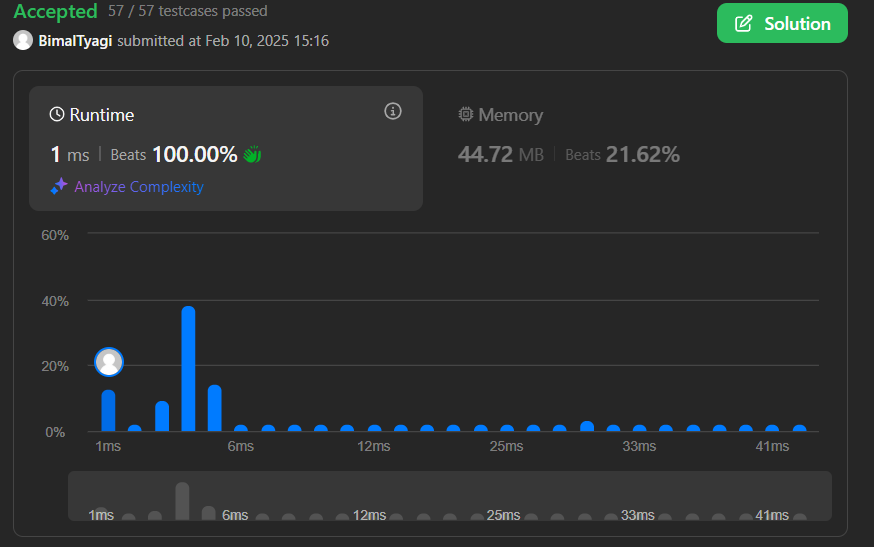
n >>= 1;

}

return ret;

}

}



1. Beautiful Array

class Solution {

public int[] beautifulArray(int n) {

int[] answer = new int[n];

if(n == 1) {

answer[0] = 1;

return answer;

}

int[] right =beautifulArray(n/2);

int[] left = beautifulArray((n+1)/2);

for(int i=left.length; i<n; i++) {

answer[i] = right[i-left.length] \* 2;

}

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

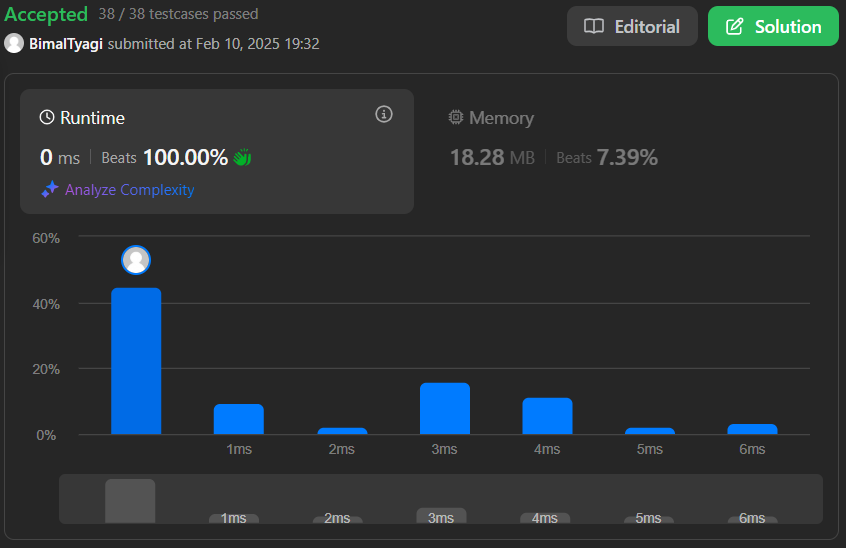
answer[i] = left[i] \* 2 - 1;

}

return answer;

}

}



1. The Skyline Problem

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

List<int[]> result = new ArrayList<>();

List<int[]> height = new ArrayList<>();

for(int[] b:buildings) {

height.add(new int[]{b[0], -b[2]});

height.add(new int[]{b[1], b[2]});

}

Collections.sort(height, (a, b) -> {

if(a[0] != b[0])

return a[0] - b[0];

return a[1] - b[1];

});

Queue<Integer> pq = new PriorityQueue<>((a, b) -> (b - a));

pq.offer(0);

int prev = 0;

for(int[] h:height) {

if(h[1] < 0) {

pq.offer(-h[1]);

} else {

pq.remove(h[1]);

}

int cur = pq.peek();

if(prev != cur) {

result.add(new int[]{h[0], cur});

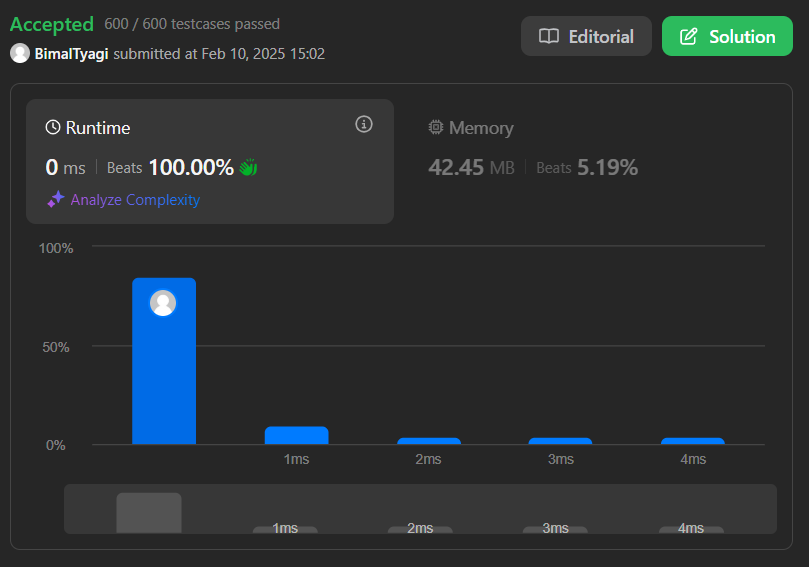
prev = cur;

}

}

return result;

}



1. 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) {

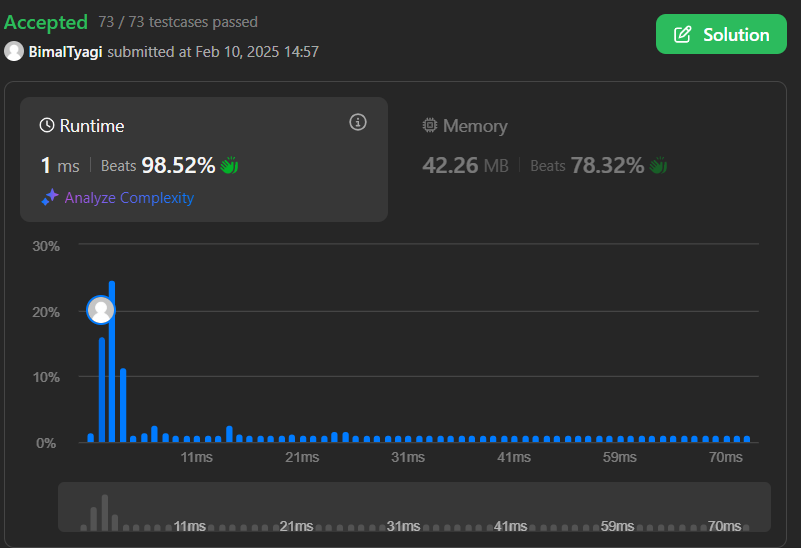
index = -(index + 1);

}

res.add(index, val);

}

}



1. 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) {

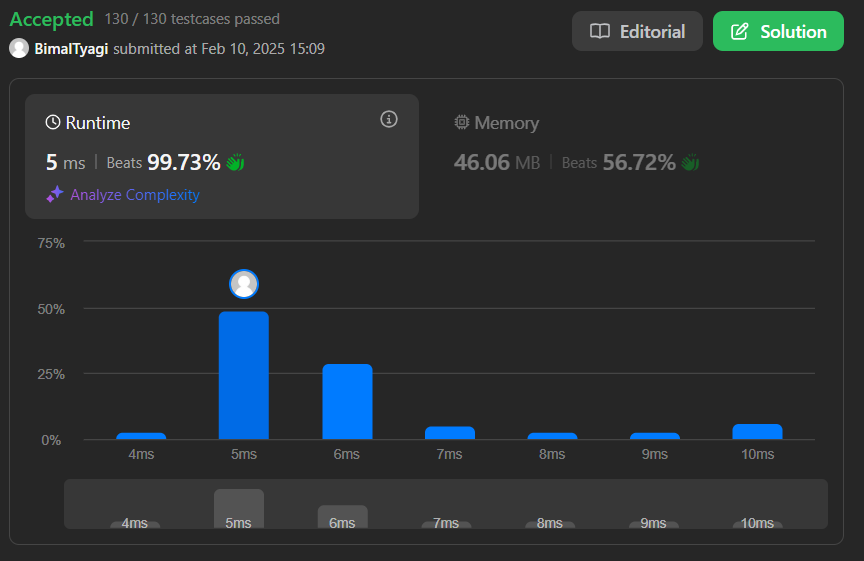
if (node.start > end || node.end < start) return 0;

if (node.start >= start && node.end <= end) return node.val;

return Math.max(rangeMaxQuery(node.left, start, end), rangeMaxQuery(node.right, start, end));

}

}



1. Merge Sorted Array

class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

int i = m - 1;

int j = n - 1;

int k = m + n - 1;

while (j >= 0) {

if (i >= 0 && nums1[i] > nums2[j]) {

nums1[k--] = nums1[i--];

} else {

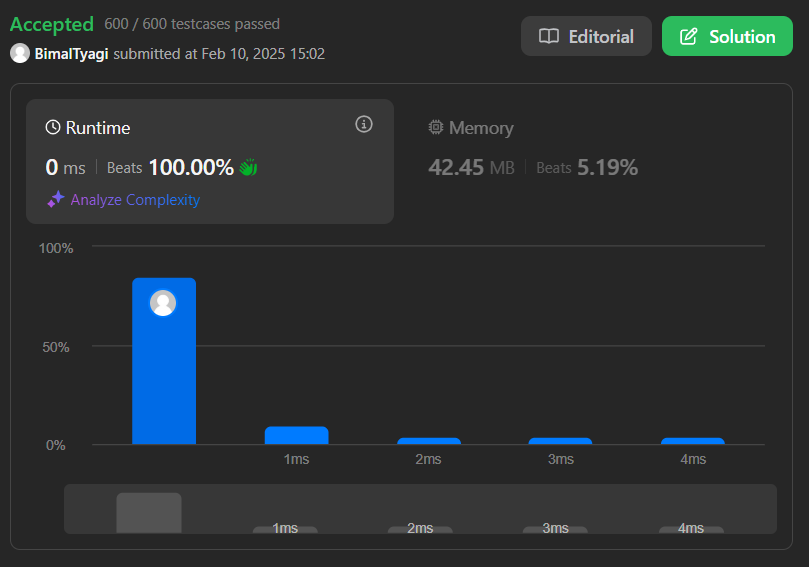
nums1[k--] = nums2[j--];

}

}

}

}



1. Sort Colors

class Solution {

public void sortColors(int[] nums) {

HashMap<Integer, Integer> count = new HashMap<>();

count.put(0, 0);

count.put(1, 0);

count.put(2, 0);

for (int num : nums) {

count.put(num, count.get(num) + 1);

}

int idx = 0;

for (int color = 0; color < 3; color++) {

int freq = count.get(color);

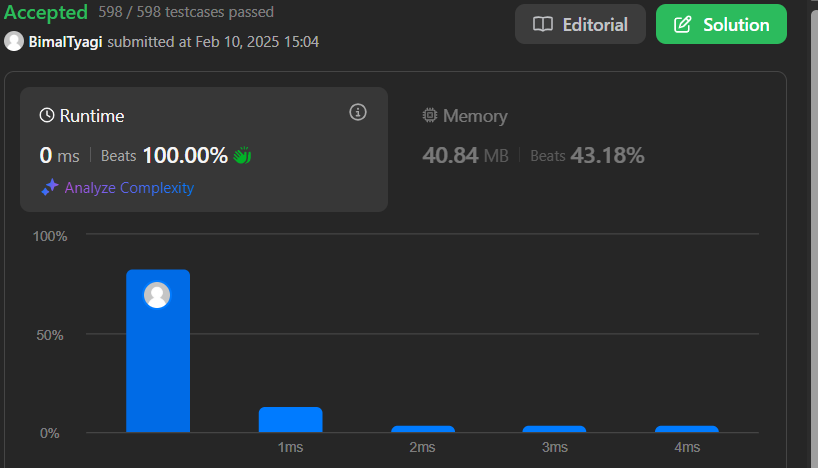
for (int j = 0; j < freq; j++) {

nums[idx] = color;

idx++;

}

}

}

}

1. Top K Frequent Elements

class Solution {

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

Map<Integer, Integer> counter = new HashMap<>();

for (int n : nums) {

counter.put(n, counter.getOrDefault(n, 0) + 1);

}

List<Integer>[] freq = new ArrayList[nums.length + 1];

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

freq[i] = new ArrayList<>();

}

for (Map.Entry<Integer, Integer> entry : counter.entrySet()) {

int frequency = entry.getValue();

freq[frequency].add(entry.getKey());

}

int[] res = new int[k];

int idx = 0;

for (int i = freq.length - 1; i >= 0; i--) {

for (int num : freq[i]) {

res[idx++] = num;

if (idx == k) {

return res;

}

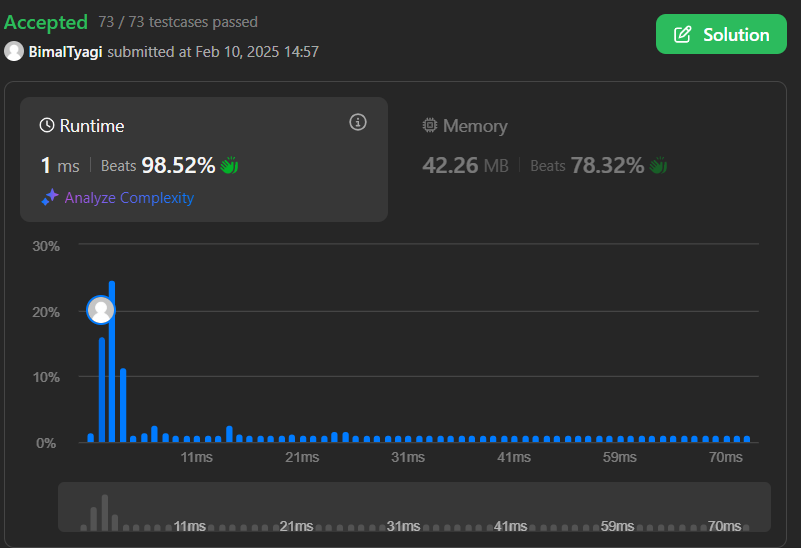
}

}

return new int[0];

}

}



1. Kth Largest Element in an array

class Solution {

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

int minValue = Arrays.stream(nums).min().getAsInt();

int maxValue = Arrays.stream(nums).max().getAsInt();

int[] count = new int[maxValue - minValue + 1];

for (int num : nums) {

count[num - minValue]++;

}

int remaining = k;

for (int i = count.length - 1; i >= 0; i--) {

remaining -= count[i];

if (remaining <= 0) {

return i + minValue;

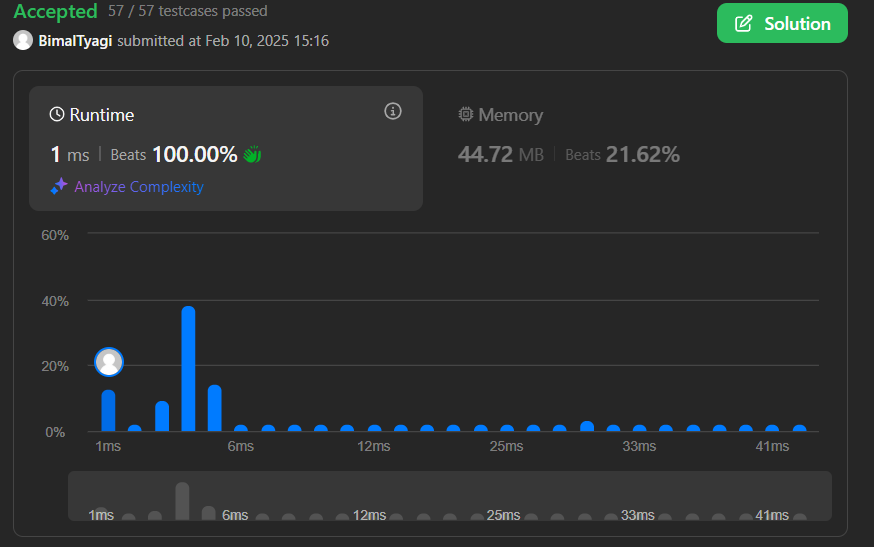
}

}

return -1; // This line should not be reached

}

}



1. Find Peak Element

class Solution {

public int findPeakElement(int[] nums) {

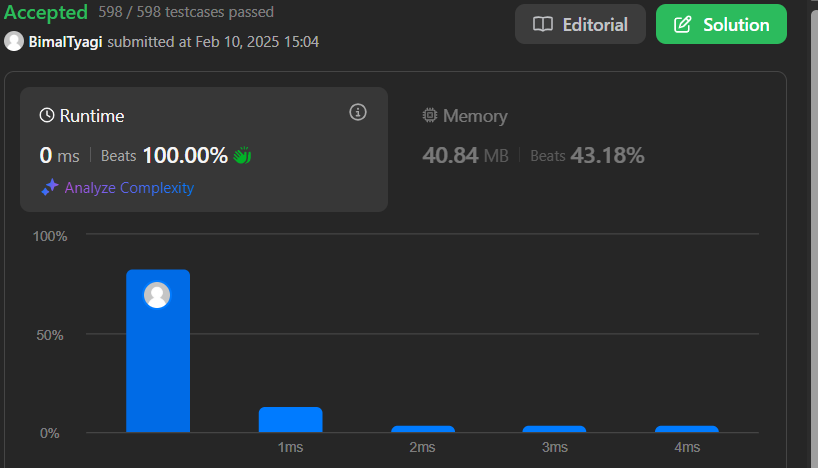
int left = 0;

int right = nums.length - 1;

while (left < right) {

int mid = (left + right) / 2;

if (nums[mid] > nums[mid + 1]) {

right = mid;

} else {

left = mid + 1;

}

}

return left;

}

}

1. Merge Intervals

class Solution {

public int[][] merge(int[][] intervals) {

Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

List<int[]> merged = new ArrayList<>();

int[] prev = intervals[0];

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

int[] interval = intervals[i];

if (interval[0] <= prev[1]) {

prev[1] = Math.max(prev[1], interval[1]);

} else {

merged.add(prev);

prev = interval;

}

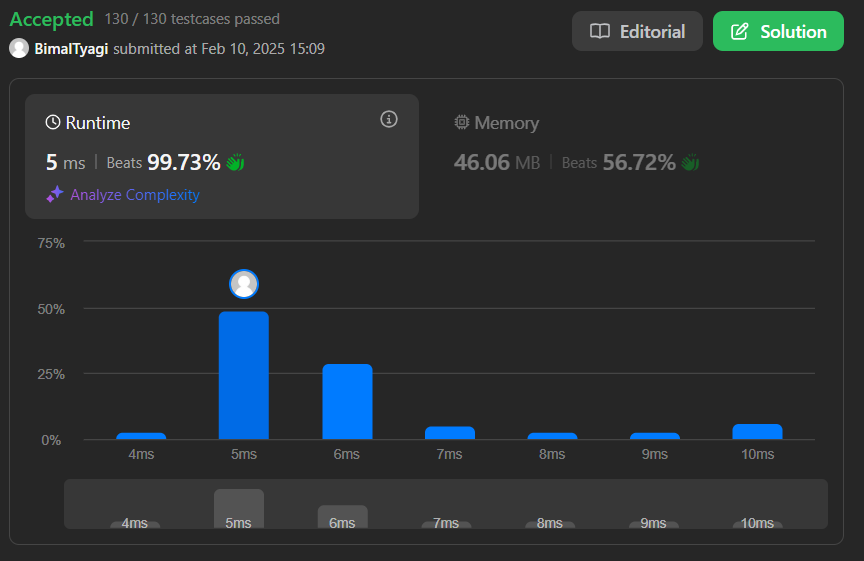
}

merged.add(prev);

return merged.toArray(new int[merged.size()][]);

}

}



1. Search in Rotated Sorted Array

class Solution {

public int search(int[] nums, int target) {

int left = 0;

int right = nums.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (nums[mid] == target) {

return mid;

} else if (nums[mid] >= nums[left]) {

if (nums[left] <= target && target <= nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (nums[mid] <= target && target <= nums[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

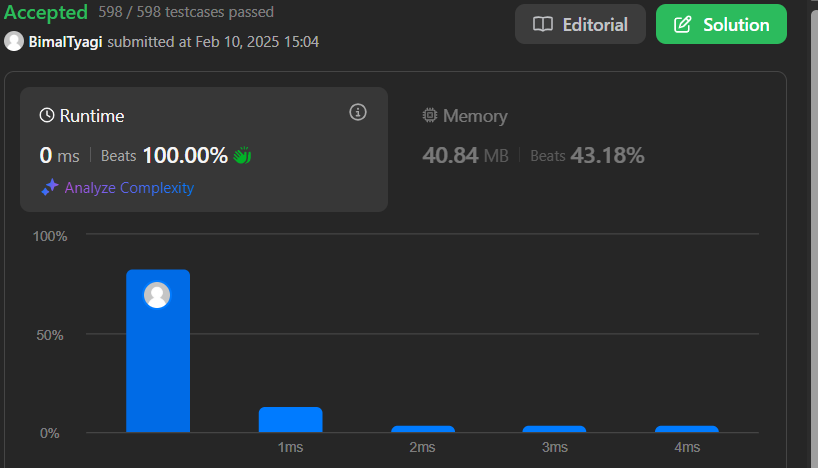
}

}

return -1;

}

}



1. Search a 2D Matrix II

class Solution {

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

int n = matrix.length, m = matrix[0].length;

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++;

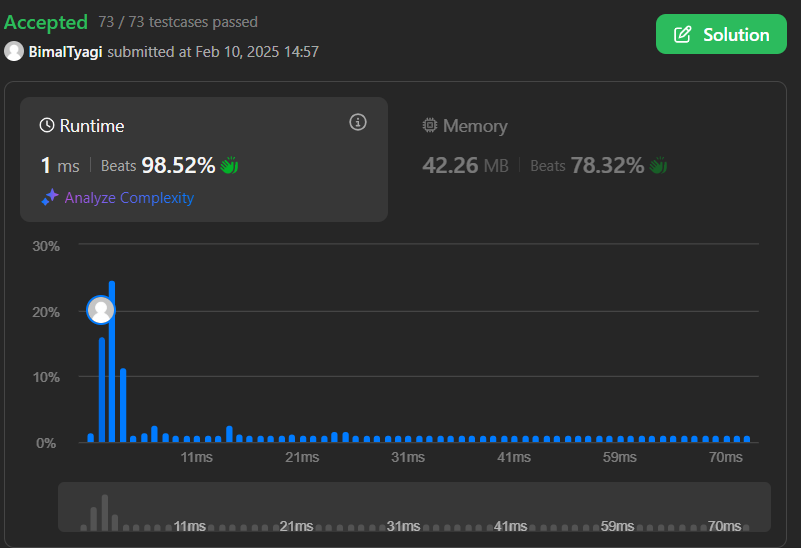
else col--;

}

return false;

}

}



1. Wiggle Sort II

class Solution {

public void wiggleSort(int[] nums) {

int n=nums.length-1;

int[] newarr=Arrays.copyOf(nums,nums.length);

Arrays.sort(newarr);

for(int i=1;i<nums.length;i+=2)

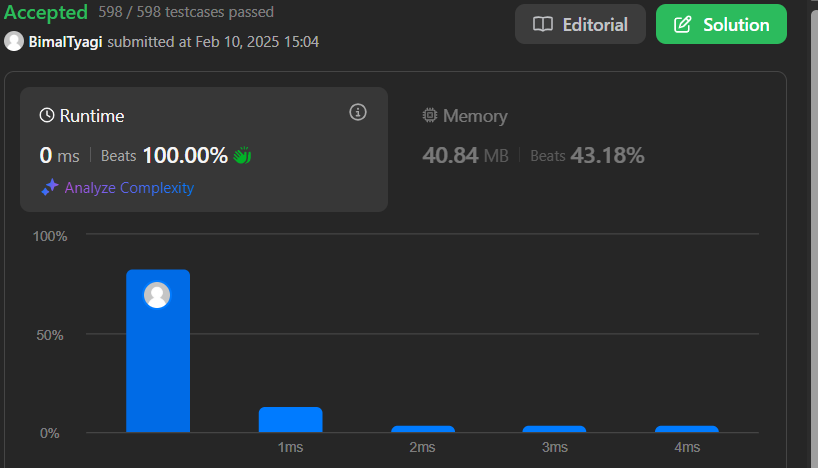
nums[i]=newarr[n--];

for(int i=0;i<nums.length;i+=2)

nums[i]=newarr[n--];

}

}



1. Kth Smallest Element in a Sorted Matrix

class Solution {

public int kthSmallest(int[][] matrix, int k) {

PriorityQueue<int[]> minHeap = new PriorityQueue<>(

(a,b) -> Integer.compare(matrix[a[0]][a[1]], matrix[b[0]][b[1]]));

for (int row = 0; row < Math.min(matrix.length, k); row++) {

minHeap.offer(new int[]{row, 0});

}

while (--k > 0) {

int[] coord = minHeap.poll();

int row = coord[0], col = coord[1];

if (col < matrix.length - 1)

minHeap.offer(new int[]{row, col + 1});

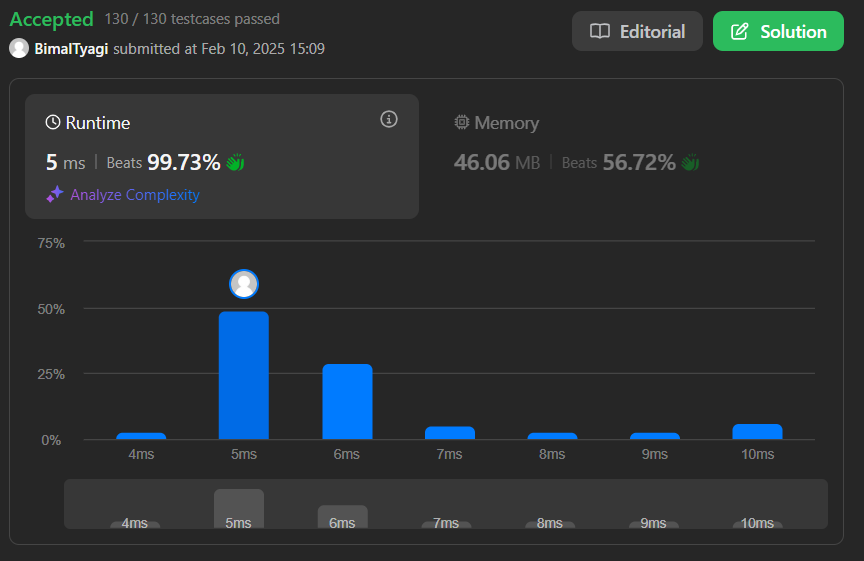
}

int[] answer = minHeap.poll();

return matrix[answer[0]][answer[1]];

}

}



1. Median of Two Sorted Arrays

class Solution

{

public double findMedianSortedArrays(int[] nums1, int[] nums2)

{

int m = nums1.length, n = nums2.length;

int[] nums = new int[m+n];

int i = 0, j = 0, ind = 0;

while(i < m || j < n)

{

if(i >= m)

{

nums[ind++] = nums2[j++];

continue;

}

else if(j >= n)

{

nums[ind++] = nums1[i++];

continue;

}

else if(i < m && j < n && nums1[i] < nums2[j])

{

nums[ind++] = nums1[i];

i++;

}

else

{

nums[ind++] = nums2[j];

j++;

}

}

int mid = (m+n)/2;

if((m+n) % 2 == 0)

{

return (double)(nums[mid] + nums[mid-1])/2;

}

return (double)nums[mid];

}

}

