Question: Convert Sorted Array to Binary Search Tree

Code:

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

return CreateBST(nums, 0, nums.length - 1);

}

private TreeNode CreateBST(int nums[], int l, int r) {

if (l > r) {

return null;

}

int mid = l + (r - l) / 2;

TreeNode root = new TreeNode(nums[mid]);

root.left = CreateBST(nums, l, mid - 1);

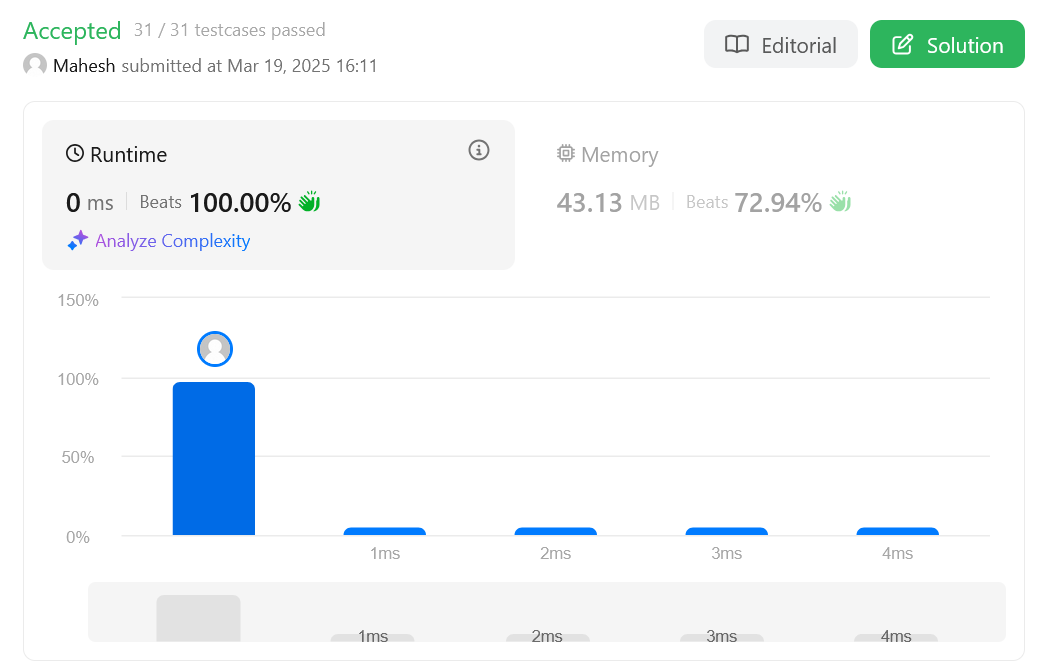
root.right = CreateBST(nums, mid + 1, r);

return root;

}

}

**Output:**



Question: Number of 1 bits

public class Solution {

public int hammingWeight(int n) {

int count = 0;

while (n != 0) {

count += (n & 1);

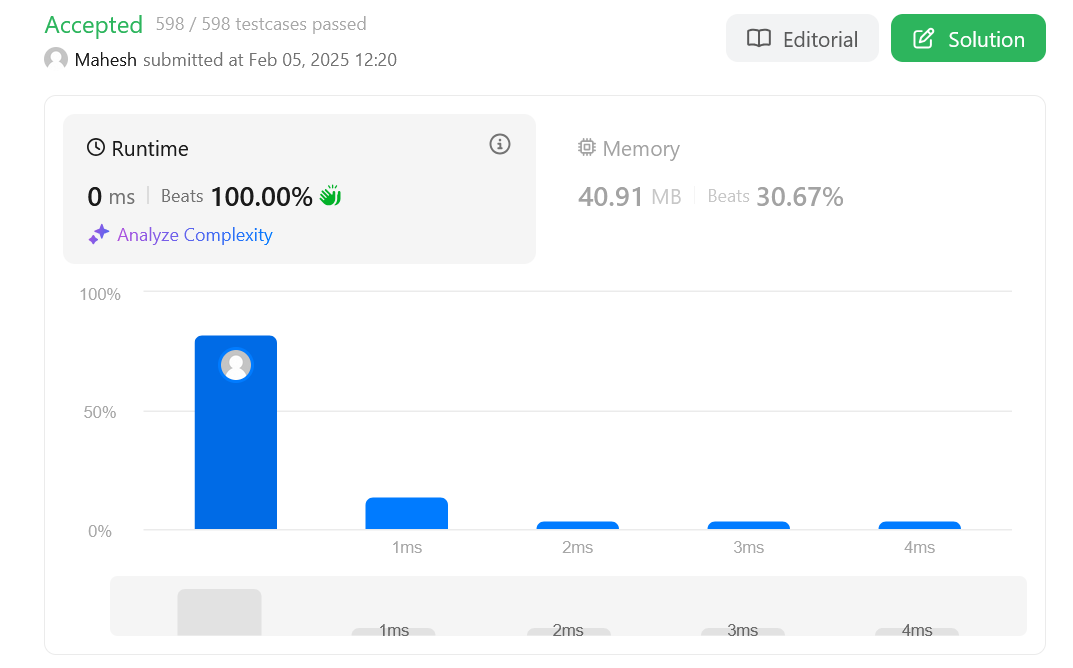
n >>>= 1;

}

return count;

}

}



Question: Sort an array

class Solution {

public int[] sortArray(int[] nums) {

mergeSort(nums, 0, nums.length - 1);

return nums;

}

private void mergeSort(int[] array, int low, int high) {

if (low >= high) {

return;

}

int mid = low + (high - low) / 2;

mergeSort(array, low, mid);

mergeSort(array, mid + 1, high);

merge(array, low, mid, high);

}

private void merge(int[] array, int low, int mid, int high) {

int n1 = mid - low + 1;

int n2 = high - mid;

int[] leftPart = new int[n1];

int[] rightPart = new int[n2];

System.arraycopy(array, low, leftPart, 0, n1);

System.arraycopy(array, mid + 1, rightPart, 0, n2);

int p1 = 0, p2 = 0, writeInd = low;

while (p1 < n1 && p2 < n2) {

if (leftPart[p1] <= rightPart[p2]) {

array[writeInd++] = leftPart[p1++];

} else {

array[writeInd++] = rightPart[p2++];

}

}

while (p1 < n1) {

array[writeInd++] = leftPart[p1++];

}

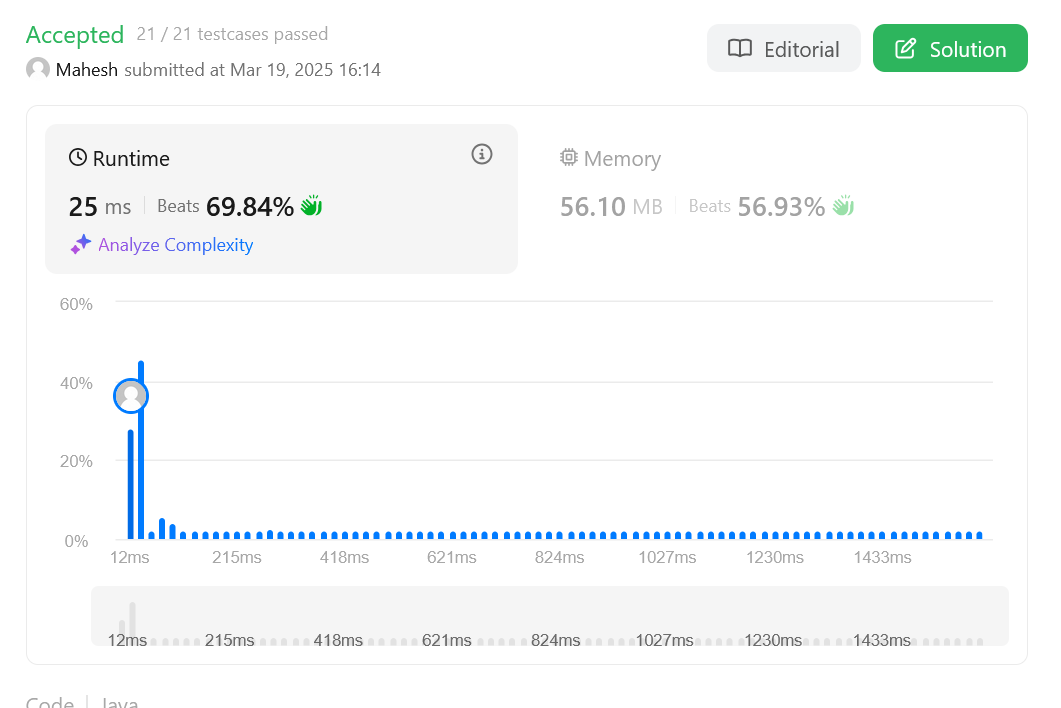
while (p2 < n2) {

array[writeInd++] = rightPart[p2++];

}

}

}



Question: Maximum Subarray

public class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0], currentSum = 0;

for (int num : nums) {

currentSum = Math.max(num, currentSum + num);

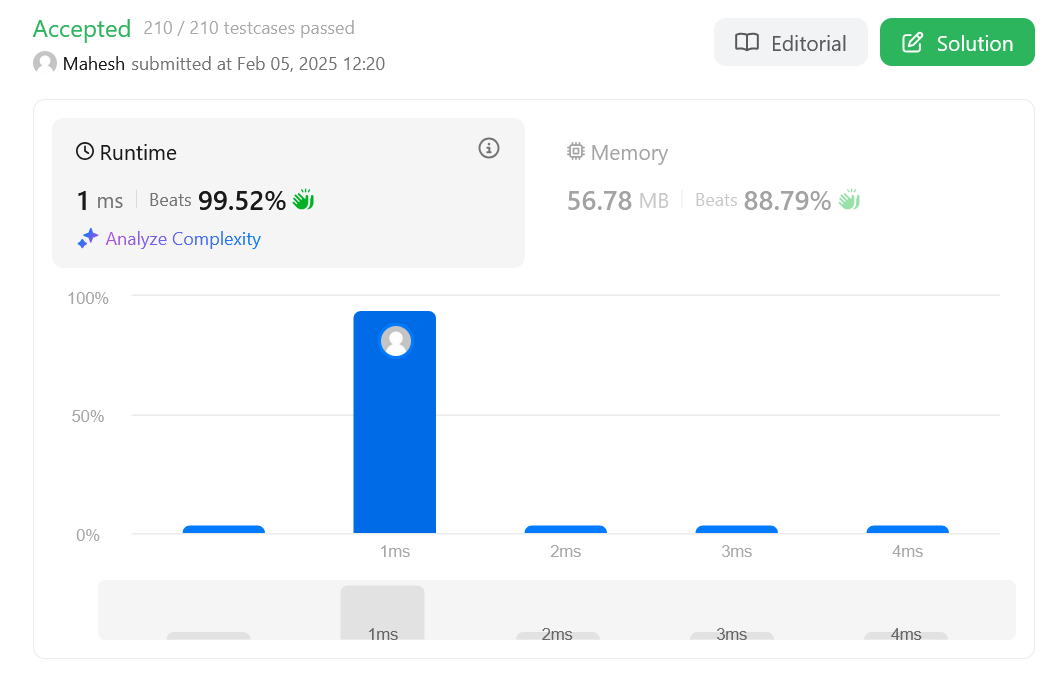
maxSum = Math.max(maxSum, currentSum);

}

return maxSum;

}

}



Question: Beautiful Array

public class Solution {

public int[] beautifulArray(int N) {

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

result.add(1);

while (result.size() < N) {

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

for (int num : result) {

if (num \* 2 - 1 <= N)

temp.add(num \* 2 - 1);

}

for (int num : result) {

if (num \* 2 <= N)

temp.add(num \* 2);

}

result = temp;

}

int[] arr = new int[result.size()];

for (int i = 0; i < result.size(); i++) {

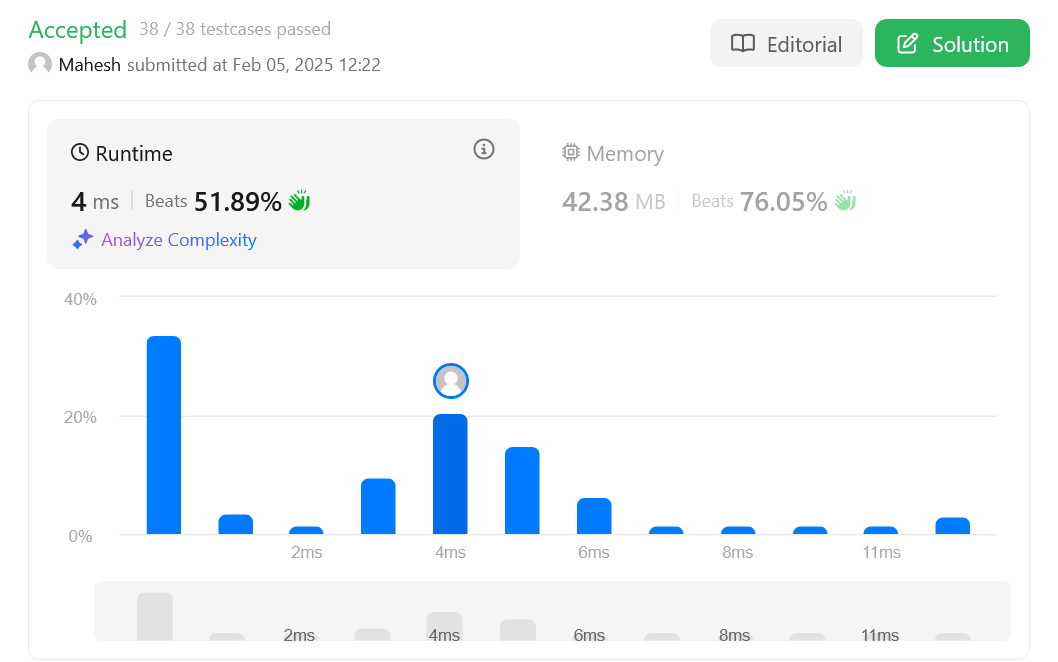
arr[i] = result.get(i);

}

return arr;

}

}



Question: Super pow

public class Solution {

private static final int MOD = 1337;

private int pow(int a, int b) {

int res = 1;

a %= MOD;

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

res = (res \* a) % MOD;

}

return res;

}

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

int res = 1;

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

res = (res \* pow(a, b[i])) % MOD;

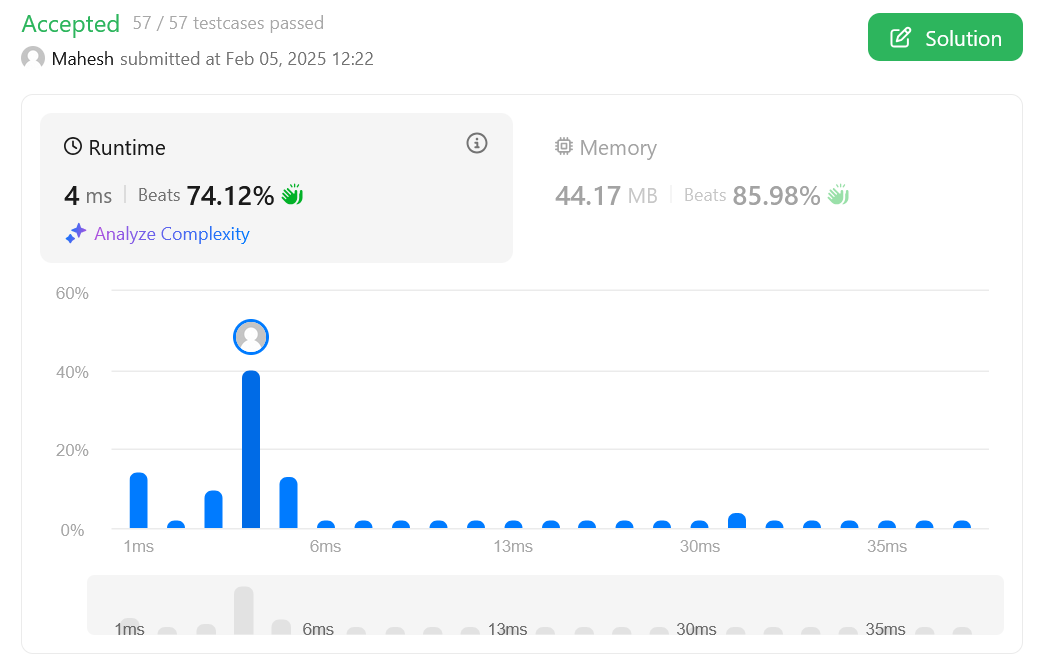
a = pow(a, 10);

}

return res;

}

}



Question: The skyline problem

class Solution {

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

return divideAndConquer(buildings, 0, buildings.length - 1);

}

private List<List<Integer>> divideAndConquer(int[][] buildings, int left, int right) {

if (left > right)

return new ArrayList<>();

if (left == right) {

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

result.add(Arrays.asList(buildings[left][0], buildings[left][2]));

result.add(Arrays.asList(buildings[left][1], 0));

return result;

}

int mid = left + (right - left) / 2;

List<List<Integer>> leftSkyline = divideAndConquer(buildings, left, mid);

List<List<Integer>> rightSkyline = divideAndConquer(buildings, mid + 1, right);

return mergeSkylines(leftSkyline, rightSkyline);

}

private List<List<Integer>> mergeSkylines(List<List<Integer>> left, List<List<Integer>> right) {

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

int h1 = 0, h2 = 0, i = 0, j = 0;

while (i < left.size() && j < right.size()) {

List<Integer> point1 = left.get(i);

List<Integer> point2 = right.get(j);

int x;

if (point1.get(0) < point2.get(0)) {

x = point1.get(0);

h1 = point1.get(1);

i++;

} else if (point1.get(0) > point2.get(0)) {

x = point2.get(0);

h2 = point2.get(1);

j++;

} else {

x = point1.get(0);

h1 = point1.get(1);

h2 = point2.get(1);

i++;

j++;

}

int maxHeight = Math.max(h1, h2);

if (result.isEmpty() || result.get(result.size() - 1).get(1) != maxHeight) {

result.add(Arrays.asList(x, maxHeight));

}

}

while (i < left.size())

result.add(left.get(i++));

while (j < right.size())

result.add(right.get(j++));

return result;

}

}

