**DSA – DAY 8 CODING PROBLEMS**

**Name:** Dhuvarrakesh K

**Date:**20/11/24

**Register Number:**22IT024

**1.3sum Closest**

**Code:**

class Solution {

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

Arrays.sort(nums);

int closestSum = 0;

int smallestDiff = Integer.MAX\_VALUE;

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

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (sum == target) {

return sum;

}

int diff = Math.abs(sum - target);

if (diff < smallestDiff) {

smallestDiff = diff;

closestSum = sum;

}

if (sum < target) {

j++;

} else {

k--;

}

}

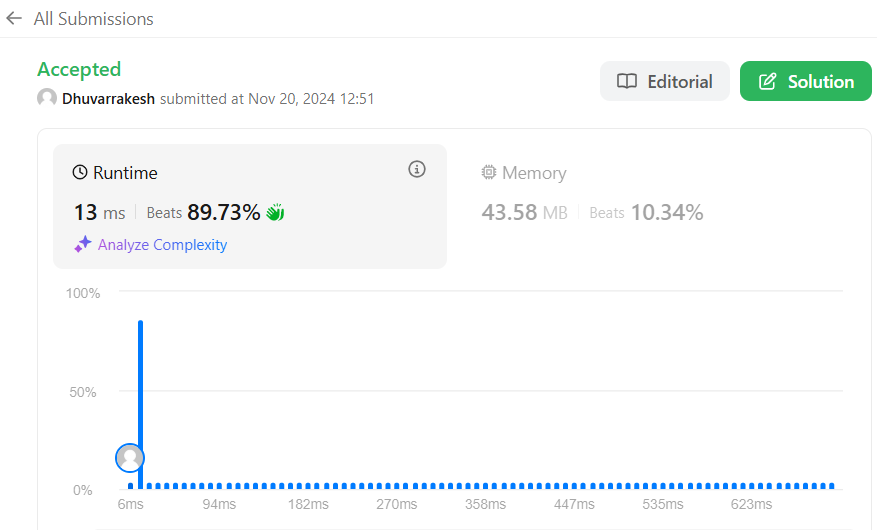
}

return closestSum;

}

}

**Output:**

****

**2.Jump Game-II**

**Code:**

class Solution {

public int jump(int[] nums) {

int jumps = 0;

int currentEnd = 0;

int farthest = 0;

for (int i = 0; i < nums.length - 1; i++) {

farthest = Math.max(farthest, i + nums[i]);

if (i == currentEnd) {

jumps++;

currentEnd = farthest;

if (currentEnd >= nums.length - 1) {

break;

}

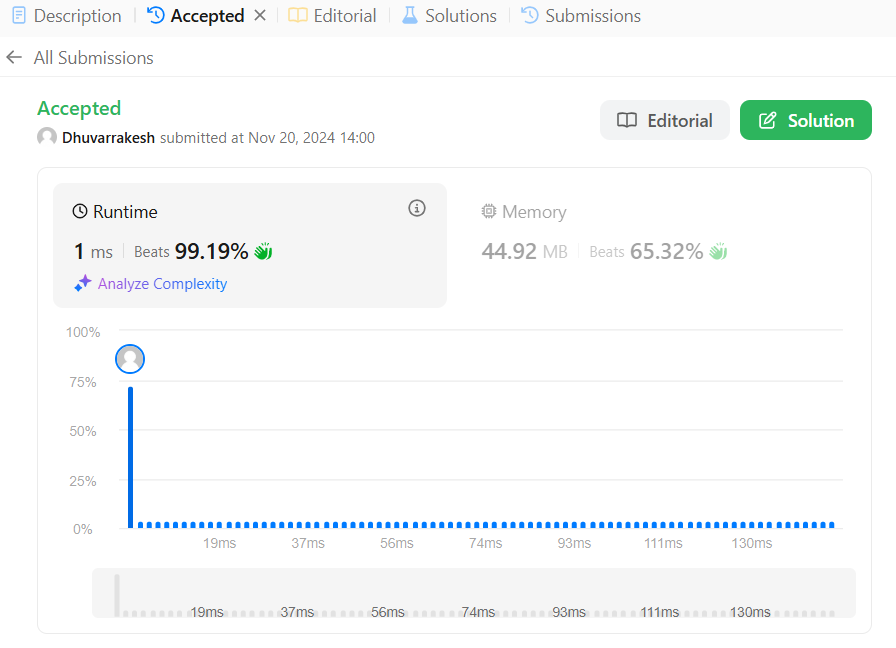
}

}

return jumps;

}

}

**Output:** ****

**3.Group Anagrams**

**Code:**

class Solution {

    public List<List<String>> groupAnagrams(String[] strs) {

        Map<String, List<String>> map = new HashMap<>();

        for (String word : strs) {

            char[] chars = word.toCharArray();

            Arrays.sort(chars);

            String sortedWord = new String(chars);

            if (!map.containsKey(sortedWord)) {

                map.put(sortedWord, new ArrayList<>());

            }

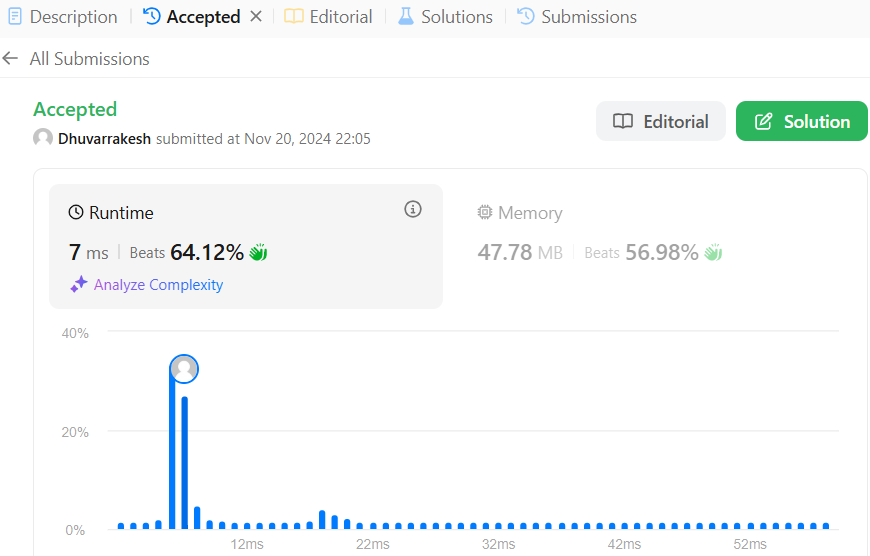
            map.get(sortedWord).add(word);

        }

        return new ArrayList<>(map.values());

    }

}

**Output:** ****

**4.Best time to buy and sell stock**

**Code:**

class Solution {

public int maxProfit(int[] prices) {

int max = 0;

int start = prices[0];

int len = prices.length;

for(int i = 1;i<len; i++){

if(start < prices[i]) max += prices[i] - start;

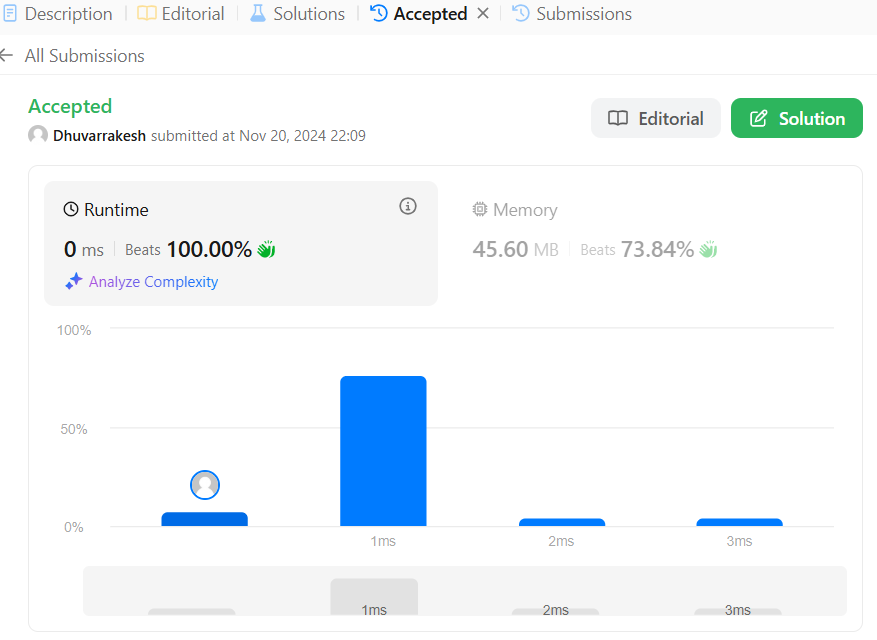
start = prices[i];

}

return max;

}

}

**Output:** ****

**5.Number of Island**

**Code:**

class Solution {

    public int numIslands(char[][] grid) {

        int islands = 0;

        int rows = grid.length;

        int cols = grid[0].length;

        Set<String> visited = new HashSet<>();

        int[][] directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

        for (int r = 0; r < rows; r++) {

            for (int c = 0; c < cols; c++) {

                if (grid[r][c] == '1' && !visited.contains(r + "," + c)) {

                    islands++;

                    bfs(grid, r, c, visited, directions, rows, cols);

                }

            }

        }

        return islands;

    }

    private void bfs(char[][] grid, int r, int c, Set<String> visited, int[][] directions, int rows, int cols) {

        Queue<int[]> q = new LinkedList<>();

        visited.add(r + "," + c);

        q.add(new int[]{r, c});

        while (!q.isEmpty()) {

            int[] point = q.poll();

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

            for (int[] direction : directions) {

                int nr = row + direction[0], nc = col + direction[1];

                if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1' && !visited.contains(nr + "," + nc)) {

                    q.add(new int[]{nr, nc});

                    visited.add(nr + "," + nc);

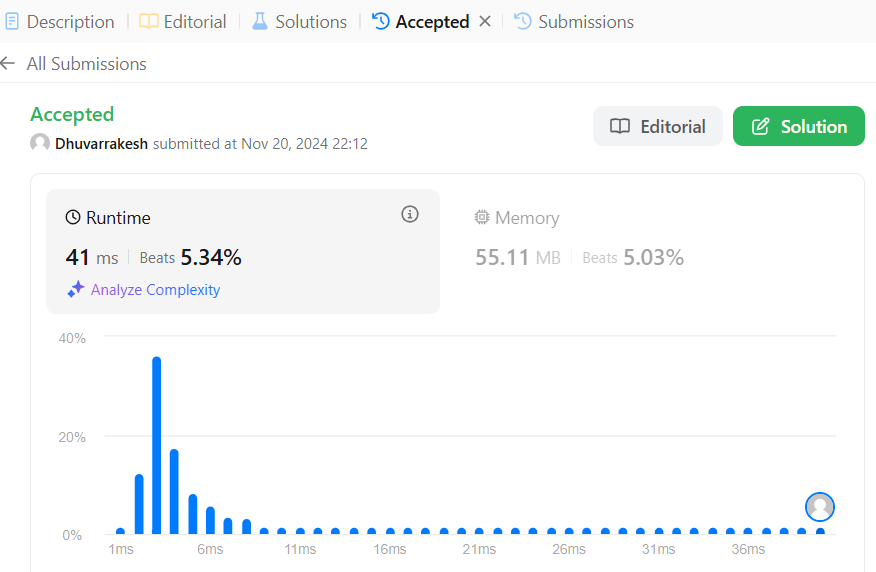
                }

            }

        }

    }

}

**Output:** ****

**6.Decode Ways**

**Code:**

class Solution {

    public int numDecodings(String s) {

        if (s.charAt(0) == '0') {

            return 0;

        }

        int n = s.length();

        int[] dp = new int[n + 1];

        dp[0] = dp[1] = 1;

        for (int i = 2; i <= n; i++) {

            int one = Character.getNumericValue(s.charAt(i - 1));

            int two = Integer.parseInt(s.substring(i - 2, i));

            if (1 <= one && one <= 9) {

                dp[i] += dp[i - 1];

            }

            if (10 <= two && two <= 26) {

                dp[i] += dp[i - 2];

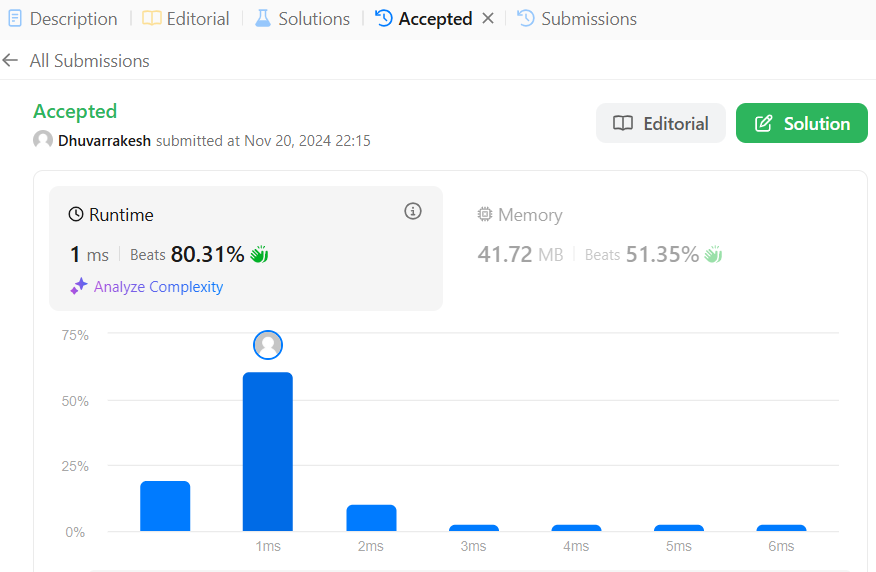
            }

        }

        return dp[n];

    }

}

**Output:** ****

**7.Merge Sort**

**Code:**

class Solution {

void mergeSort(int arr[], int l, int r) {

if (l < r) {

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[l + i];

for (int i = 0; i < n2; i++)

R[i] = arr[m + 1 + i];

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

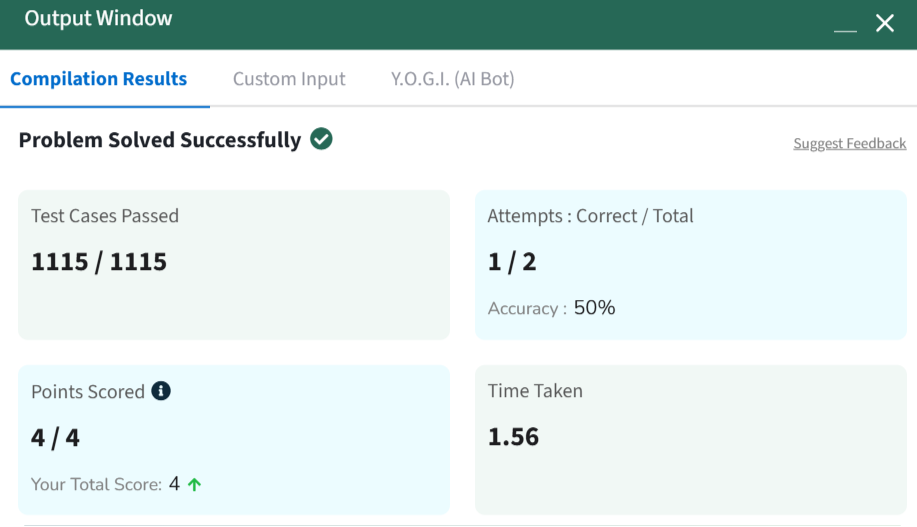
j++;

k++;

}

}

}

**Output:** ****

**8.Quick Sort**

**Code:**

class Solution {

static void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

static int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

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

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

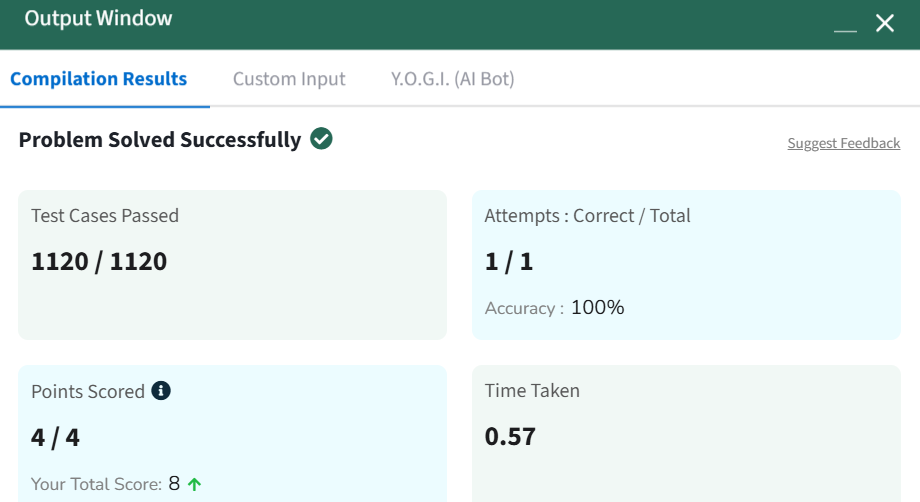
arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

}

**Output:** ****

**9.Ternary Search**

**Code:**

class Solution {

static int ternarySearch(int arr[], int N, int K) {

int low = 0, high = N - 1;

while (high >= low) {

int mid1 = low + (high - low) / 3;

int mid2 = high - (high - low) / 3;

if (arr[mid1] == K) return 1;

if (arr[mid2] == K) return 1;

if (K < arr[mid1]) high = mid1 - 1;

else if (K > arr[mid2]) low = mid2 + 1;

else {

low = mid1 + 1;

high = mid2 - 1;

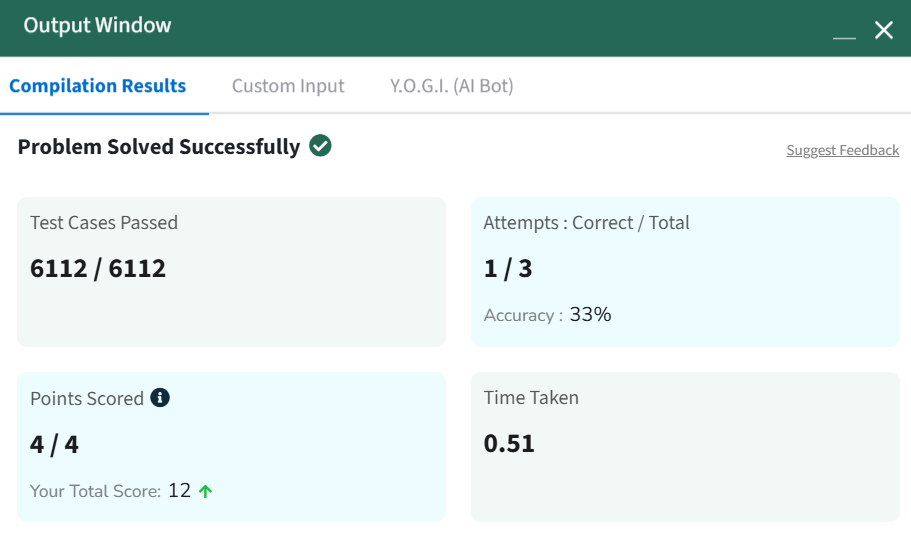
}

}

return -1;

}

}

**Output:** ****