DAY2

CODILITY

**1.BINARY GAP**

class Solution {

public int solution(int N) {

String binaryString = Integer.toBinaryString(N);

int maxGap = 0;

int start = -1;

int end = -1;

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

if (binaryString.charAt(i) == '1') {

if (start != -1) {

end = i;

maxGap = Math.max(maxGap, end - start - 1);

start = i;

} else {

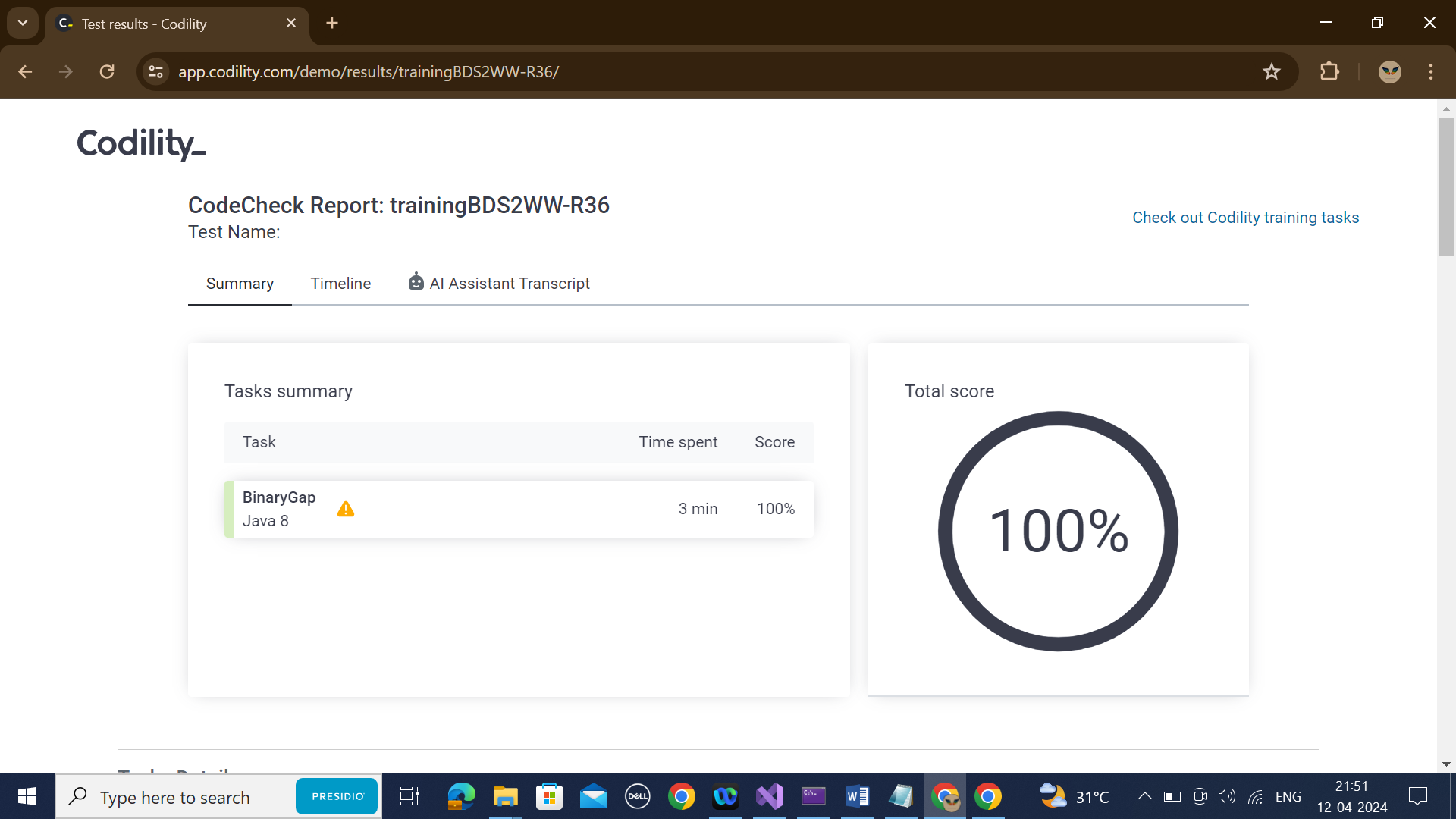
start = i;

}

}

}

return maxGap;

} 

**2.CYCLIC ROTATION**

import java.util.Arrays;

public class Solution {

public int[] solution(int[] A, int K) {

int N = A.length;

int[] result = new int[N];

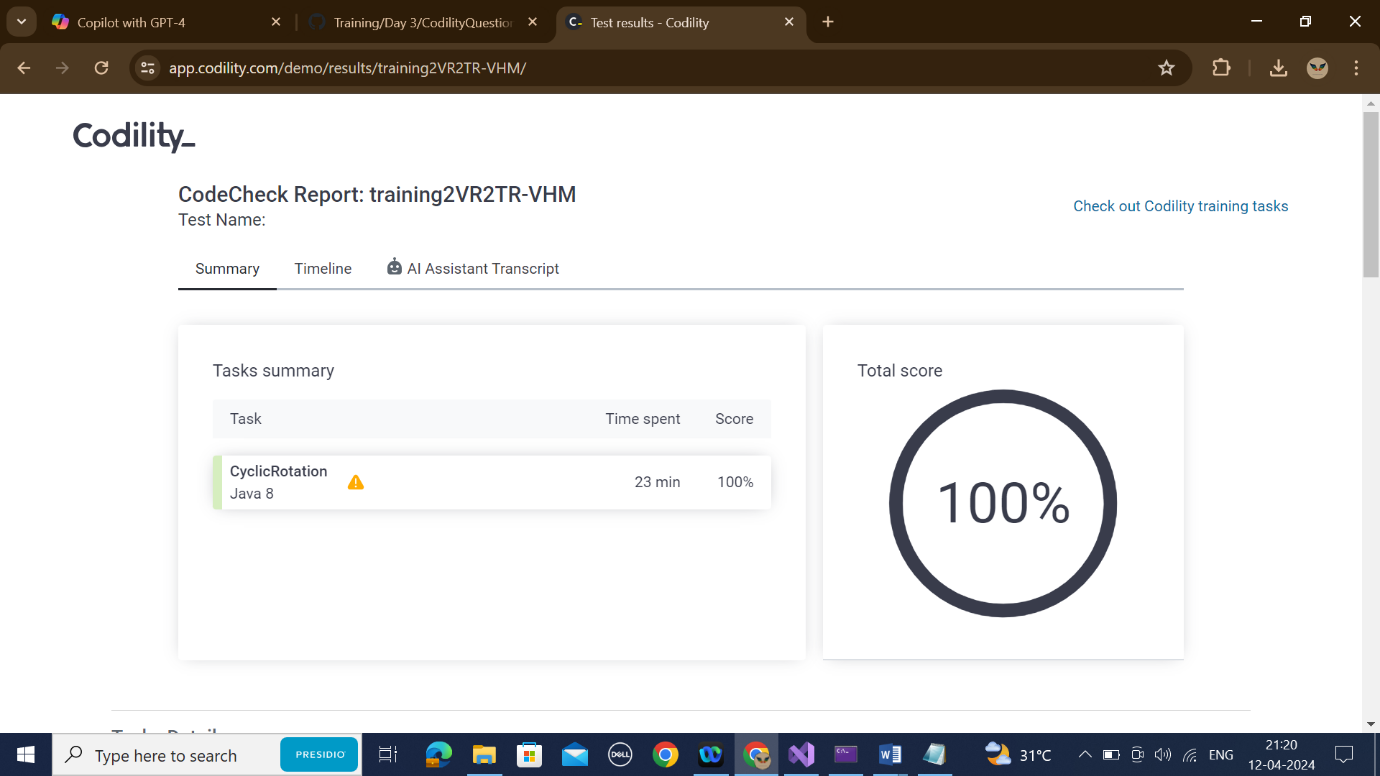
for (int i = 0; i < N; i++) {

result[(i + K) % N] = A[i];

}

return result;

}

}

**3.FROG JUMP**

class Solution {

public int solution(int X, int Y, int D) {

int distance = Y - X;

if (distance % D == 0) {

return distance / D;

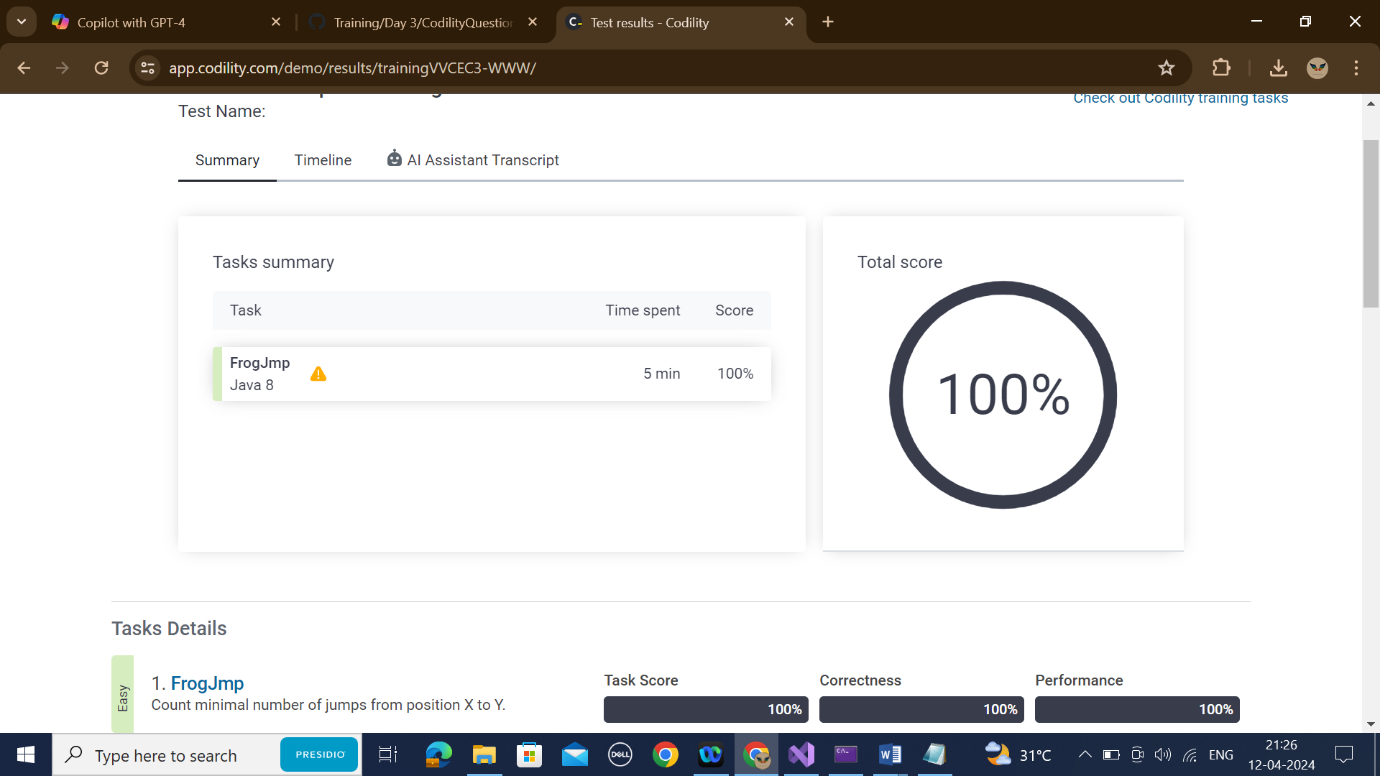
} else {

return distance / D + 1;

}

}

}



**4.ODD OCCURANCES IN AN ARRAY**

class Solution {

public int solution(int[] A) {

int res = 0;

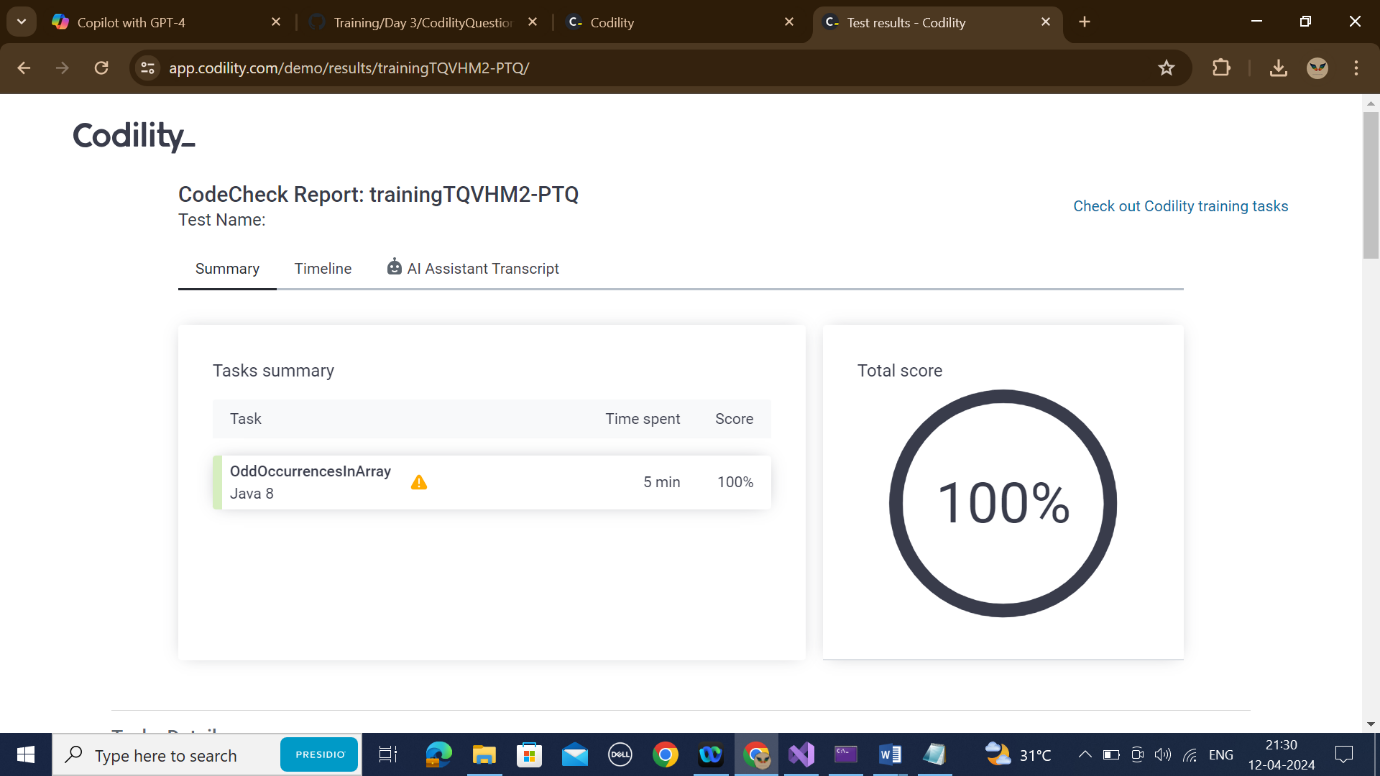
for (int i = 0; i < A.length; i++) {

res = res ^ A[i];

}

return res;

}

}

**5.PERMUTATION MISSING ELEMENT**

**CODE:**

class Solution {

public int solution(int[] A) {

long N = A.length;

long totalSum = (N + 1) \* (N + 2) /2;

long arraySum = 0;

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

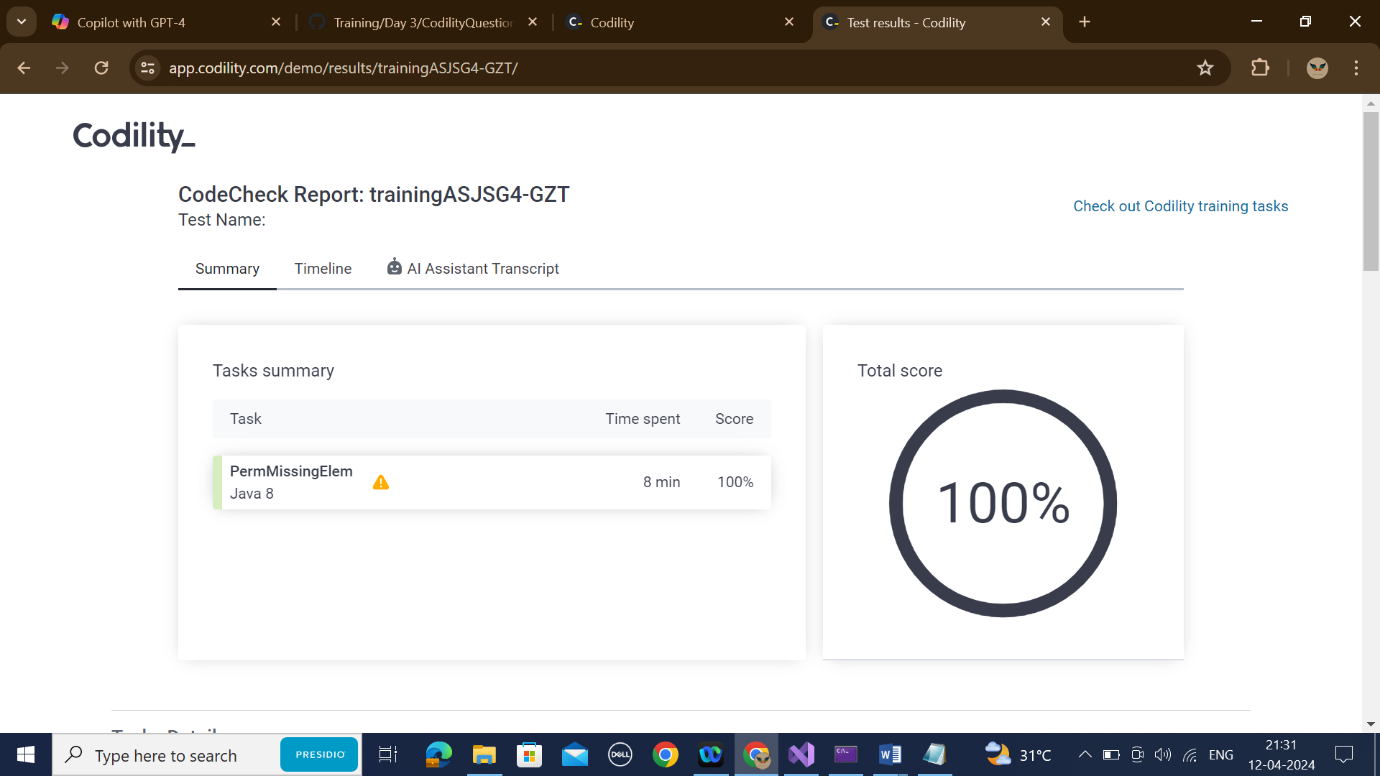
arraySum += A[i];

}

return (int)(totalSum - arraySum);

}

}

****

**6.TAPE EQUILIBRIUM:**

**CODE:**

public class Solution {

public int solution(int[] A) {

long sumOfAll = 0;

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

sumOfAll += A[i];

}

long sumOfFirstPart = 0;

long minDifference = Long.MAX\_VALUE;

for(int p=0; p<A.length-1; p++) {

sumOfFirstPart += A[p];

long sumOfSecondPart = sumOfAll - sumOfFirstPart;

long difference = Math.abs(sumOfFirstPart - sumOfSecondPart);

if(difference < minDifference) {

minDifference = difference;

}

}

return (int)minDifference}}

OUTPUT:

