Harsh Sukhachari Bhagat

Test ID: 450010654000057 | 9970422026 | harshbhagat9970@gmail.com

Test Date: March 25, 2025

Automata

98 /100

Automata

98 / 100

Programming Ability	Programming Practices	Functional Correctness
100 / 100	88 / 100	100 / 100

1 | Introduction

About the Report

This report provides a detailed analysis of the candidate's performance on different assessments. The tests for this job role were decided based on job analysis, O*Net taxonomy mapping and/or criterion validity studies. The candidate’s responses to these tests help construct a profile that reflects her/his likely performance level and achievement potential in the job role

This report has the following sections:

The **Summary** section provides an overall snapshot of the candidate’s performance. It includes a graphical representation of the test scores and the subsection scores.

The **Response** section captures the response provided by the candidate. This section includes only those tests that require a subjective input from the candidate and are scored based on artificial intelligence and machine learning.

Score Interpretation

All the test scores are on a scale of 0-100. All the tests except personality and behavioural evaluation provide absolute scores. The personality and behavioural tests provide a norm-referenced score and hence, are percentile scores. Throughout the report, the colour codes used are as follows:

- 🟢 Scores between 67 and 100
- 🟡 Scores between 33 and 67
- 🔴 Scores between 0 and 33

2 | Response

Automata ● 98 / 100 Code Replay

Question 1 (Language: Java)

In a science research lab, combining two nuclear chemicals produces a maximum energy that is the product of the energy of the two chemicals. The energy values of the chemicals can be negative or positive. The scientist wants to calculate the sum of the energies of the two chemicals which produces maximum energy on reaction.

Write an algorithm to find the sum of the energy of the two chemicals which produces maximum energy on reaction.

Scores

Programming Ability

100 / 100

Completely correct. A correct implementation of the problem using the right control-structures and data dependencies.

Programming Practices

75 / 100

Low readability, high on program structure. The source code does not follow best practices in its formatting and may contain a few redundant/improper coding constructs.

Functional Correctness

100 / 100

Functionally correct source code. Passes all the test cases in the test suite for a given problem.

Final Code Submitted — Compilation Status: Pass

```
1  import java.util.*;
2  import java.lang.*;
3  import java.io.*;
4  
5  /*
6   *
7   */
8  public class Solution
9  {
10     public static int  maxEnergy(int[] ener)
11     {
12         if(ener.length < 2){
13             return 0;
14         }
15 
16         int  answer = 0;
17         // Write your code here
```

Code Analysis

Average-case Time Complexity

Candidate code: O(N logN)

Best case code: O(log N)

*N represents number of chemicals.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

Readability & Language Best Practices

Line 21,22,23...: Variables are given very short names.

```java

    Arrays.sort(ener);

    int n = ener.length;
    int p2 = ener[n - 1] * ener[n - 2];
    int p1 = ener[0] * ener[1];

    if(p2 >= p1){
      answer =  ener[n - 1] + ener[n - 2];
    }

    else {
      answer = ener[0] + ener[1];
    }

    return answer;
  }

  public static void main(String[] args)
  {
    Scanner in = new Scanner(System.in);
    //input for ener
    int ener_size = in.nextInt();
    int ener[] = new int[ener_size];
    for(int idx = 0; idx < ener_size; idx++)
    {
      ener[idx] = in.nextInt();
    }

    int result = maxEnergy(ener);
    System.out.print(result);

  }
}

```

Test Case Execution Passed TC: **100%**

Total score 13/13

100%	**100%**	**100%**
Basic(**8**/8)	Advance(**4**/4)	Edge(**1**/1)

Compilation Statistics

2	2	0	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:04:17**
Average time taken between two compile attempts:	**00:02:09**
Average test case pass percentage per compile:	**100%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 2 (Language: Java)

Charlie has a magic mirror that shows the right-rotated versions of a given word. To generate different right rotations of a word, the word is written in a circle in clockwise order and read starting from any given character in clockwise order until all the characters are covered. For example, in the word "*sample*", if we start with '*p*', we get the right rotated word as "*plesam*".

Write an algorithm to output 1 if the *word1* is a right rotation of *word2* otherwise output -1.

Scores

Programming Ability

100 / 100

Completely correct. A correct implementation of the problem using the right control-structures and data dependencies.

Programming Practices

100 / 100

High readability, high on program structure. The source code is readable and does not consist of any significant redundant/improper coding constructs.

Functional Correctness

100 / 100

Functionally correct source code. Passes all the test cases in the test suite for a given problem.

Final Code Submitted

Compilation Status: Pass

```
import java.util.*;
import java.lang.*;
import java.io.*;

/*
 * word1, represents the first word.
word2, represents the second word.
 */
public class Solution
{
   public static int  isSameReflection(String word1, String word2)
   {
       if(word1.equals(word2)){
          return 1;
       }

       else if(word1.length() != word2.length()){
          return -1;
       }

      int  answer = 0;
      // Write your code here
      String str = word1 + word1;              // logic
      str = str.toLowerCase();
      if(str.contains(word2)){
         answer = 1;
      }

      else {
         answer = -1;
      }

```

Code Analysis

Average-case Time Complexity

Candidate code: O(N)

Best case code: O(N)

*N represents maximum of length of both the input strings

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

```
        return answer;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        // input for word1
        String word1 = in.nextLine();

        // input for word2
        String word2 = in.nextLine();

        int result = isSameReflection(word1, word2);
        System.out.print(result);

    }
}

```

Test Case Execution

Passed TC: **100%**

Total score

18/18

100%
Basic(**8**/8)

100%
Advance(**8**/8)

100%
Edge(**2**/2)

Compilation Statistics

16	16	0	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:40:11**
Average time taken between two compile attempts:	**00:02:31**
Average test case pass percentage per compile:	**79.86%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code