Harsh Sukhachari Bhagat

Test ID: 450010344000057 | 9970422026 | harshbhagat9970@gmail.com

Test Date: February 25, 2025

Automata

100 /100

Automata

100 / 100

Programming Ability	Programming Practices	Functional Correctness
100 / 100	100 / 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 ● 100 / 100 | Code Replay

Question 1 (Language: Java)

An e-commerce company is planning to give a special discount on all its products to its customers for the holiday. The company possesses data on its stock of N product types. The data for each product type represents the count of customers who have ordered the given product. If the data K is positive then the product has been ordered by K customers and is in stock. If the data K is negative then the product has been ordered by K customers but is not in stock. The company will fulfill the order directly if the ordered product is in stock. If it is not in stock then the company will fulfill the order after they replenish the stock from the warehouse. They are planning to offer a discount amount A for each product. The discount value will be distributed to the customers who have purchased that selected product. The discount will be distributed only if the discount amount A can be divided by the number of orders for a particular product.

Write an algorithm for the sales team to find the number of products out of N for which the discount will be distributed.

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.*;

/*
 *
 */
public class Solution
{
    public static int  noOfProducts(int[] order, int disAmount)
```

Code Analysis

Average-case Time Complexity

Candidate code: O(N)

Best case code: O(N)

*N represents number of different types of products.

Errors/Warnings

```
  {
    if(order.length == 0){
      return 0;
    }

    int  answer = 0;
    for(int i =0; i<order.length;i++){
      if(disAmount % order[i] == 0 && order[i] > 0){   //  logic
        answer++;
      }
    }
    return answer;
  }

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


    int result = noOfProducts(order, disAmount);
    System.out.print(result);

  }
}
```

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score 11/11

100%	**100%**	**100%**
Basic(**6**/6)	Advance(**2**/2)	Edge(**3**/3)

Compilation Statistics

12	12	0	0	0	1
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:27:05**
Average time taken between two compile attempts:	**00:02:15**
Average test case pass percentage per compile:	**93.18%**

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)

You are given a list of numbers. Write an algorithm to remove all the duplicate numbers of the list so that the list contains only distinct numbers in the same order as they appear in the input list.

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

```
1  import java.util.*;
2  import java.lang.*;
3  import java.io.*;
4
5  /*
6   * arr, representing the list of positive integers.
7   */
8  public class Solution
9  {
10     public static int[]  removeDuplicate(int[] arr)
11     {
12
13         // Write your code here
14         if(arr.length == 0){
15             return new int [] {0};
16         }
17
18         if(arr.length == 1){
19             return new int [] { arr[0] };
20         }
21
22         // HashSet for avoid Duplicates elements
23         LinkedHashSet<Integer> set = new LinkedHashSet<>();
24         for(int i : arr){
25             set.add(i);
26         }
27
28         int answer [] = new int[set.size()];  // use array of size equivalent of hashset.
29         int index = 0;
30
31         for(int nums : set){
```

Code Analysis

Average-case Time Complexity

Candidate code: O(N logN)

Best case code: O(N)

*N represents number of elements in the input list.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

Readability & Language Best Practices

Line 24: Variables are given very short names.

```
            answer[index] = nums;
            index++;
        }
        return answer;
    }

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

        int[] result = removeDuplicate(arr);
        for(int idx = 0; idx < result.length - 1; idx++)
        {
            System.out.print(result[idx] + " ");
        }
        System.out.print(result[result.length - 1]);
    }
}
```

Test Case Execution

Passed TC: **100%**

Total score 20/20

100%	100%	100%
Basic(**10**/10)	Advance(**7**/7)	Edge(**3**/3)

Compilation Statistics

14	12	2	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:15:25**
Average time taken between two compile attempts:	**00:01:06**
Average test case pass percentage per compile:	**54.64%**

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