

Harsh Bhagat

Test ID: 432007404241059 | 9970422026 | harshbhagat.23d@stvincentngp.edu.in

Test Date: October 22, 2024

Automata

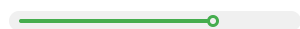
68 /100



Automata

68 / 100

Programming Ability



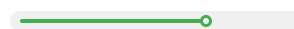
70 / 100

Programming Practices



50 / 100

Functional Correctness



67 / 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



68 / 100

[Code Replay](#)

Question 1 (Language: Java)

A cold storage company has N storage units for various products. The company has received N orders that must be preserved at respective N temperatures inside the storage units. The company manager wants to identify which products must be preserved at negative temperatures.

Write an algorithm to help the manager find the number of products that have negative temperature storage requirements.

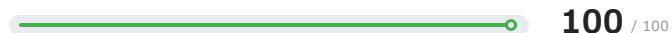
Scores

Programming Ability



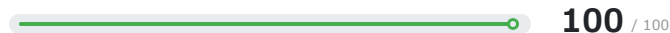
Completely correct. A correct implementation of the problem using the right control-structures and data dependencies.

Functional Correctness



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

Programming Practices



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

Final Code Submitted

Compilation Status: Pass

```

1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
5 /*
6  * "temperature" representing the temperatures at which the prod
7  * ucts must be preserved in the storage units.
8  */
9 public class Solution
10 {
11     public static int productsAtNegativeTemp(int[] temperature)
12     {
13         int answer = 0;
14
15         if(temperature.length==0){
16             answer = 0;
17         }
18     }
19 }
```

Code Analysis

Average-case Time Complexity

Candidate code: $O(N)$

Best case code: $O(N)$

*N represents number of products.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

```

16  }
17
18  for(int i=0;i<temparature.length;i++){
19      if(temparature[i]<0){
20          answer++;
21      }
22  }
23
24  return answer;
25  }
26
27  public static void main(String[] args)
28  {
29      Scanner in = new Scanner(System.in);
30      //input for temparature
31      int temparature_size = in.nextInt();
32      int temparature[] = new int[temparature_size];
33      for(int idx = 0; idx < temparature_size; idx++)
34      {
35          temparature[idx] = in.nextInt();
36      }
37
38      int result = productsAtNegativeTemp(temparature);
39      System.out.print(result);
40
41  }
42 }
43

```

Test Case Execution

Passed TC: 100%

Total score

13/13

100%

Basic(8/8)

100%

Advance(4/4)

100%

Edge(1/1)

Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:05:53

Average time taken between two compile attempts:

00:02:57

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)

An agent sends a message to headquarters containing the details of his project. He sends one soft copy to the agency's computer (P) and sends one hard copy by fax to Roger, the technical head of the agency (Q). During the transmission noise in the network causes some bits of the data message P to get distorted. However, we know that Roger always matches the binary values of both messages and checks whether he can convert the message P to message Q by flipping the minimum number of bits.

Write an algorithm to help Roger find the minimum number of bits that must be flipped to convert message P to message Q.

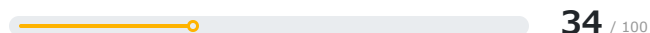
Scores

Programming Ability



Emerging basic structure. Appropriate keywords and tokens present, showing some understanding of a part of the problem.

Functional Correctness



Partially correct basic functionality. The source code compiles and passes only some of the basic test cases. Some advanced or edge cases may randomly pass.

Programming Practices



Programming practices score cannot be generated. This is because source code has syntax/runtime errors and is unparseable or the source code does not meet the minimum code-length specifications.

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 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 flippedBits(int num1, int num2) 11 { 12 int answer = 0; 13 14 String bin1 = Integer.toBinaryString(num1); 15 String bin2 = Integer.toBinaryString(num2); 16 for(char c: bin1.toCharArray()){ 17 if(c == '1'){ 18 answer++; 19 } 20 } 21 22 return answer; 23 } 24 25 public static void main(String[] args) 26 { 27 Scanner in = new Scanner(System.in); 28 // input for num1 29 int num1 = in.nextInt(); 30 31 // input for num2 32 int num2 = in.nextInt(); 33 34 35 int result = flippedBits(num1, num2); 36 System.out.print(result); 37 38 } 39 } 40 </pre>		<p>Average-case Time Complexity</p> <p>Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p>Best case code: O(1)</p> <p>*N represents constant</p> <p>Errors/Warnings</p> <p>There are no errors in the candidate's code.</p> <p>Structural Vulnerabilites and Errors</p> <p>There are no errors in the candidate's code.</p>

Test Case Execution	Passed TC: 38.46%		
<p>Total score</p> <div> <div></div> <div>5/13</div> </div>	<p>67%</p> <p>Basic(4/6)</p>	<p>0%</p> <p>Advance(0/6)</p>	<p>100%</p> <p>Edge(1/1)</p>

Compilation Statistics

10

Total attempts

9

Successful

1

Compilation errors

0

Sample failed

0

Timed out

3

Runtime errors

Response time:

00:37:22

Average time taken between two compile attempts:

00:03:44

Average test case pass percentage per compile:

27.69%

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