

Rutu Hinge

Test ID: 432007530001038 | 7798224336 | rutu.hinge23@vit.edu

Test Date: October 19, 2024

Logical Ability 65 /100	Quantitative Ability (Advanced) 61 /100	English Comprehension 78 /100	Automata Fix 15 /100
Automata Pro 69 /100	Personality Completed		

Logical Ability			65 / 100
Inductive Reasoning	Deductive Reasoning	Abductive Reasoning	
63 / 100	73 / 100	58 / 100	

Quantitative Ability (Advanced)			61 / 100
Basic Mathematics	Advanced Mathematics	Applied Mathematics	
62 / 100	63 / 100	59 / 100	

English Comprehension		78 / 100	CEFR: C1
Grammar	Vocabulary	Comprehension	
86 / 100	71 / 100	78 / 100	

Automata Fix

15 / 100

Syntactical Error

100 / 100

Logical Error

0 / 100

Code Reuse

0 / 100

Automata Pro

69 / 100

Programming Ability

70 / 100

Programming Practices

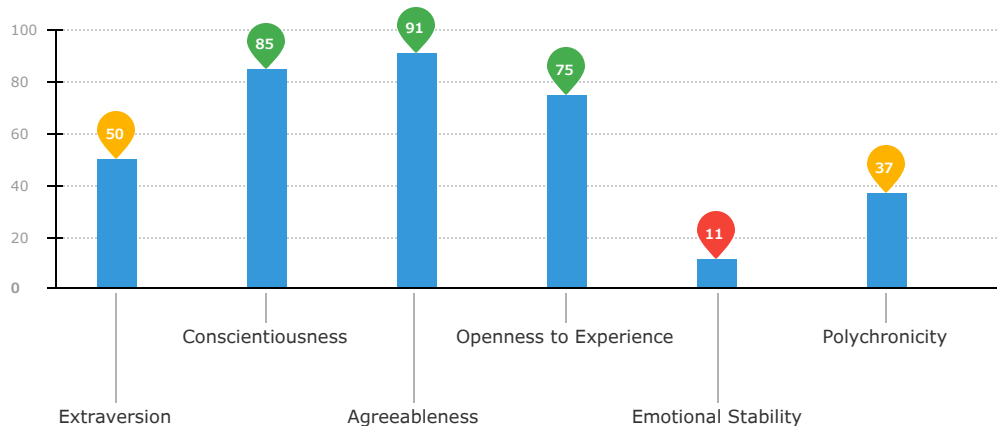
75 / 100

Functional Correctness

61 / 100

Personality

Completed



Competencies

People Interaction

Self-Drive

Trainability

Repetitive Job Suitability

Work attributes

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 **Insights** section provides detailed feedback on the candidate's performance in each of the tests. The descriptive feedback includes the competency definitions, the topics covered in the test, and a note on the level of the candidate's performance.

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.

The **Learning Resources** section provides online and offline resources to improve the candidate's knowledge, abilities, and skills in the different areas on which s/he was evaluated.

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 | Insights

English Comprehension

78 / 100

CEFR: **C1**

This test aims to measure your vocabulary, grammar and reading comprehension skills.

You have a fairly rich vocabulary and a strong command of English grammar. You are able to read and understand complex text. Having a good command of the English language is important to communicate with internal stakeholders and clients, as well as to interpret reports, articles and complex texts at work.

Logical Ability

65 / 100



Inductive Reasoning

63 / 100

This competency aims to measure the your ability to synthesize information and derive conclusions.

It is commendable that you have excellent inductive reasoning skills. You are able to make specific observations to generalize situations and also formulate new generic rules from variable data.



Deductive Reasoning

73 / 100

This competency aims to measure the your ability to synthesize information and derive conclusions.

It is commendable that you have excellent inductive reasoning skills. You are able to make specific observations to generalize situations and also formulate new generic rules from variable data.



Abductive Reasoning

58 / 100

Quantitative Ability (Advanced)

61 / 100

This test aims to measure your ability to solve problems on basic arithmetic operations, probability, permutations and combinations, and other advanced concepts.

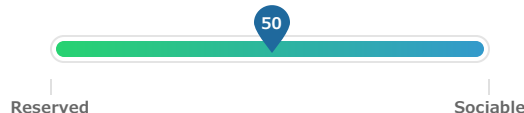
You are able to solve word problems on basic concepts of percentages, ratio, proportion, interest, time and work. Having a strong hold on these concepts can help you understand the concept of work efficiency and how interest is accrued on bank savings. It can also guide you in time management, work planning, and resource allocation in complex projects.

Personality

Competencies



Extraversion



Extraversion refers to a person's inclination to prefer social interaction over spending time alone. Individuals with high levels of extraversion are perceived to be outgoing, warm and socially confident.

- You are comfortable socializing to a certain extent. You prefer small gatherings in familiar environments.
- You feel at ease interacting with your close friends but may be reserved among strangers.
- You indulge in activities involving thrill and excitement that are not too risky.
- You contemplate the consequences before expressing any opinion or taking an action.
- You take charge when the situation calls for it and you are comfortable following instructions as well.
- Your personality may be suitable for jobs demanding flexibility in terms of working well with a team as well as individually.



Conscientiousness



Conscientiousness is the tendency to be organized, hard working and responsible in one's approach to your work. Individuals with high levels of this personality trait are more likely to be ambitious and tend to be goal-oriented and focused.

- You value order and self discipline and tends to pursue ambitious endeavours.
- You believe in the importance of structure and is very well-organized.
- You carefully review facts before arriving at conclusions or making decisions based on them.
- You strictly adhere to rules and carefully consider the situation before making decisions.
- You tend to have a high level of self confidence and do not doubt your abilities.
- You generally set and work toward goals, try to exceed expectations and are likely to excel in most jobs, especially those which require careful or meticulous approach.



Agreeableness



Agreeableness refers to an individual's tendency to be cooperative with others and it defines your approach to interpersonal relationships. People with high levels of this personality trait tend to be more considerate of people around them and are more likely to work effectively in a team.

- You are considerate and sensitive to the needs of others.
- You tend to put the needs of others ahead of your own.
- You are likely to trust others easily without doubting their intentions.
- You are compassionate and may be strongly affected by the plight of both friends and strangers.
- You are humble and modest and prefer not to talk about personal accomplishments.

- Your personality is more suitable for jobs demanding cooperation among employees.



Openness to Experience



Openness to experience refers to a person's inclination to explore beyond conventional boundaries in different aspects of life. Individuals with high levels of this personality trait tend to be more curious, creative and innovative in nature.

- You tend to be curious in nature and is generally open to trying new things outside your comfort zone.
- You may have a different approach to solving conventional problems and tend to experiment with those solutions.
- You are creative and tends to appreciate different forms of art.
- You are likely to be in touch with your emotions and is quite expressive.
- Your personality is more suited for jobs requiring creativity and an innovative approach to problem solving.



Emotional Stability

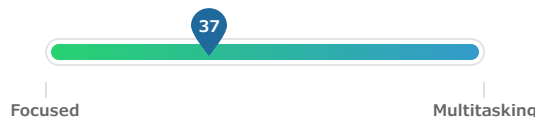


Emotional stability refers to the ability to withstand stress, handle adversity, and remain calm and composed when working through challenging situations. People with high levels of this personality trait tend to be more in control of their emotions and are likely to perform consistently despite difficult or unfavourable conditions.

- You are likely to be sensitive, emotional and may tend to worry about situations.
- You may react to everyday events with greater intensity and may become emotional.
- You may hesitate to face certain stressful situations and might feel anxious about your ability to handle them.
- You may find it hard to elicit self restraint and may tend to make impulsive decisions.
- Your personality is more suited for less stressful jobs.



Polychronicity



Polychronicity refers to a person's inclination to multitask. It is the extent to which the person prefers to engage in more than one task at a time and believes that such an approach is highly productive. While this trait describes the personality disposition of a person to multitask, it does not gauge their ability to do so successfully.

- You neither have a strong preference nor dislike to perform multiple tasks simultaneously.
- You are open to both options - pursuing multiple tasks at the same time or working on a single project at a time.
- Whether or not you will succeed in a polychronous environment depends largely on your ability to do so.

3 | Response

Automata Pro

69 / 100
[Code Replay](#)

Question 1 (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.

Functional Correctness

100 / 100

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

Programming Practices

50 / 100

High readability, low on program structure. The source code contains redundant/improper coding constructs and a few readability and formatting issues.

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 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 int[] answer = new int[100]; 13 int currentIndex = 0; 14 15 for(int i = 0; i < arr.length; i++){ 16 boolean isDuplicate = false; 17 for(int j = 0; j < currentIndex; j++){ 18 if(arr[i] == answer[j]){ 19 isDuplicate = true; 20 break; </pre>		<div>Average-case Time Complexity</div> <div> Candidate code: $O(N \log N)$ Best case code: $O(N)$ *N represents number of elements in the input list. </div> <div>Errors/Warnings</div> <div>There are no errors in the candidate's code.</div> <div>Structural Vulnerabilites and Errors</div> <div> Performance & Correctness Line 28: System.arraycopy is more efficient </div>


```

21     }
22     }
23     if(!isDuplicate){
24         answer[currentIndex++] = arr[i];
25     }
26 }
27 int[] result = new int[currentIndex];
28 for(int i = 0; i < currentIndex; i++){
29     result[i] = answer[i];
30 }
31
32 return result;
33 }
34
35 public static void main(String[] args)
36 {
37     Scanner in = new Scanner(System.in);
38     //input for arr
39     int arr_size = in.nextInt();
40     int arr[] = new int[arr_size];
41     for(int idx = 0; idx < arr_size; idx++)
42     {
43         arr[idx] = in.nextInt();
44     }
45
46     int[] result = removeDuplicate(arr);
47     for(int idx = 0; idx < result.length - 1; idx++)
48     {
49         System.out.print(result[idx] + " ");
50     }
51     System.out.print(result[result.length - 1]);
52 }
53 }
54

```

Test Case Execution

Passed TC: 100%

Total score

 20/20

100%

Basic(10/10)

100%

Advance(7/7)

100%

Edge(3/3)

Compilation Statistics

3

Total attempts

1

Successful

2

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:37:08

Average time taken between two compile attempts:

00:12:23

Average test case pass percentage per compile:

33.33%

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

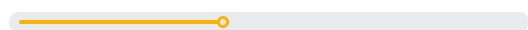
Question 2 (Language: Java)

You are performing a science experiment in a research laboratory. You are attempting to form a new compound. A compound is made up of molecules, and the mass of the compound is the sum of the masses of the molecules that compose the compound. For this experiment you have identified four types of molecules: A, B, C and D. From these four molecules, A and B are monatomic, but C and D are diatomic. A monoatomic molecule is made up of one atom, but a diatomic molecule is made up of two atoms. So the mass of a diatomic molecule is twice its atomic mass while the mass of a monoatomic molecule is equal to its atomic mass. You have to form a compound X of mass Q using the maximum number of molecules.

Write an algorithm to find the maximum number of molecules that can be used to form compound X.

Scores

Programming Ability

 **40** / 100

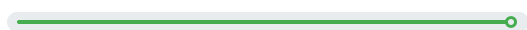
Emerging basic structure. Appropriate keywords and tokens present, showing some understanding of a part of the problem.

Functional Correctness

 **22** / 100

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

 **100** / 100

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  * atomicMassA, atomicMassB are the atomic masses of atom A and
7  * atom B, respectively.
8  * atomicMassC, atomicMassD are the atomic masses of atom C and
9  * atom D, respectively.
10 * atomicMassX is the atomic mass of compound X.
11 */
12 public class Solution
13 {
14     public static int numMolecules(int atomicMassA, int atomicMass
15     B, int atomicMassC, int atomicMassD, int atomicMassX)
16     {
17         int answer = 0;
18         for(int i = 0; i <= atomicMassX / atomicMassA; i++){
19             for(int j = 0; j <= atomicMassX / atomicMassB; j++){
20                 for(int k = 0; k <= atomicMassX / atomicMassC; k++){
21                     for(int l = 0; l <= atomicMassX / atomicMassD; l++){
22                         if(i * atomicMassA + j * atomicMassB + k * atomicMas
23                         sC + l * atomicMassD == atomicMassX){
24                             int total = i + j + k + l;
25                             if(total > answer){
26                                 answer = total;
27                             }
28                         }
29                     }
30                 }
31             }
32         }
33     }
34 }

```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code: $O(N)$

*N represents mass of molecules

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilities and Errors

There are no errors in the candidate's code.

```

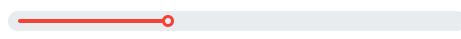
29
30
31     return answer;
32 }
33
34 public static void main(String[] args)
35 {
36     Scanner in = new Scanner(System.in);
37     // input for atomicMassA
38     int atomicMassA = in.nextInt();
39
40     // input for atomicMassB
41     int atomicMassB = in.nextInt();
42
43     // input for atomicMassC
44     int atomicMassC = in.nextInt();
45
46     // input for atomicMassD
47     int atomicMassD = in.nextInt();
48
49     // input for atomicMassX
50     int atomicMassX = in.nextInt();
51
52
53     int result = numMolecules(atomicMassA, atomicMassB, atomic
54     MassC, atomicMassD, atomicMassX);
55     System.out.print(result);
56 }
57 }
58

```

Test Case Execution

Passed TC: 33.33%

Total score

 6/18

14%

Basic(1/7)

29%

Advance(2/7)

75%

Edge(3/4)

Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

0

Sample failed

2

Timed out

0

Runtime errors

Response time:

00:14:41

Average time taken between two compile attempts:

00:07:21

Average test case pass percentage per compile:

33.33%

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

Automata Fix



15 / 100

[Code Replay](#)

Question 1 (Language: Java)

The function/method ***multiplyNumber*** returns an integer representing the multiplicative product of the maximum two of three input numbers. The function/method ***multiplyNumber*** accepts three integers- *numA*, *numB* and *numC*, representing the input numbers.

The function/method ***multiplyNumber*** compiles unsuccessfully due to syntactical error. Your task is to debug the code so that it passes all the test cases.

Scores

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 import java.lang.Math; 2 class Solution 3 { 4 int multiplyNumber(int numA, int numB, int numC) 5 { 6 int result,min,max,mid; 7 max = Math.max(numA, Math.max(numB, numC)); 8 min = Math.min(numA, Math.min(numB, numC)); 9 mid = numA + numB + numC - max - min; 10 result= max * mid; 11 return result; 12 } 13 } 14 15 </pre>		Average-case Time Complexity <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:</p> <p>*N represents</p>
		Errors/Warnings <p>There are no errors in the candidate's code.</p>
		Structural Vulnerabilites and Errors <p>There are no errors in the candidate's code.</p>

Test Case Execution	Passed TC: 100%		
Total score 10/10	100% Basic(7/7)	100% Advance(3/3)	0% Edge(0/0)

Compilation Statistics					
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors
Response time:					00:05:45
Average time taken between two compile attempts:					00:02:53
Average test case pass percentage per compile:					50%

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)

The function/method ***sameElementCount*** returns an integer representing the number of elements of the input list which are even numbers and equal to the element to its right. For example, if the input list is [4 4 4 1 8 4 1 1 2 2] then the function/method should return the output '3' as it has three similar groups i.e, (4, 4), (4, 4), (2, 2).

The function/method ***sameElementCount*** accepts two arguments - *size*, an integer representing the size of the input list and *inputList*, a list of integers representing the input list.

The function/method compiles successfully but fails to return the desired result for some test cases due to incorrect implementation of the function/method ***sameElementCount***. Your task is to fix the code so that it passes all the test cases.

Note:

In a list, an element at index *i* is considered to be on the left of index *i+1* and to the right of index *i-1*. The last element of the input list does not have any element next to it which makes it incapable to satisfy the second condition and hence should not be counted.

Scores

Final Code Submitted

Compilation Status: Pass

```
1 import java.util.*;
2 class Solution
3 {
4     int sameElementCount(int size, int[] inputList)
5     {
6         int count =0;
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

```

7   for(int i=0;i<size-1;i++)
8   {
9       if(inputList[i] == inputList[i + 1]){
10          count++;
11      }
12  }
13  return count;
14  }
15 }

```

Best case code:

*N represents

Errors/Warnings

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: **37.5%**

Total score

3/8

29%
Basic(2/7)

0%
Advance(0/0)

100%
Edge(1/1)

Compilation Statistics

2

Total attempts

1

Successful

1

Compilation errors

1

Sample failed

0

Timed out

0

Runtime errors

Response time:	00:03:29
Average time taken between two compile attempts:	00:01:45
Average test case pass percentage per compile:	6.3%

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 3 (Language: Java)

You are given a predefined structure/class **Point** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method **isRightTriangle** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.

The function/method **isRightTriangle** accepts three points - *P1*, *P2*, *P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method **isRightTriangle** so that it passes all test cases.

Helper Description

The following class is used to represent a point and is already implemented in the default code (Do not write this definition again in your code):

```
public class Point
{
    public int x, y;

    public Point()
    { }

    public Point(int x1, int y1)
    {
        x = x1 ;
        y = y1 ;
    }
}
```

```

        y = y',
    }

    public double calculateDistance(Point P)
    {
        /*Return the euclidean distance between two input points.

        This can be called as -

        * If P1 and P2 are two points then -

        * P1.calculateDistance(P2);*/
    }
}

```

Scores

Final Code Submitted	Compilation Status: Fail	Code Analysis
<pre> 1 import java.util.*; 2 class Point{ 3 public int x,y; 4 public Point(int x, int y){ 5 this.x = x; 6 this.y = y; 7 } 8 } 9 public class Solution 10 { 11 public int isRightTriangle(Point P1, Point P2, Point P3) 12 { 13 int d1 = dist(P1, P2); 14 int d2 = dist(P2, P3); 15 int d3 = dist(P1, P3); 16 return(d1 + d2 == d3 d1 + d3 == d2 d2 + d3 == d1) ? 1 : 0; 17 } 18 private int dist(Point A, Point B){ 19 return (A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y - B.y); 20 } 21 } 22 23 </pre>		Average-case Time Complexity
		<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:</p> <p>*N represents</p>
		Errors/Warnings
		<p>Solution.java:2: error: duplicate class: Point</p> <pre> class Point{ ^ 1 error </pre>
		Structural Vulnerabilites and Errors
		There are no errors in the candidate's code.

Compilation Statistics

6

Total attempts

0

Successful

6

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:09:44

Average time taken between two compile attempts:

00:01:37

Average test case pass percentage per compile:

0%

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 4 (Language: Java)

The function/method **manchester** print space-separated integers with the following property: for each element in the input list *arr*, if the bit *arr[i]* is the same as *arr[i-1]*, then the element of the output list is 0. If they are different, then its 1. For the first bit in the input list, assume its previous bit to be 0. This encoding is stored in a new list.

The function/method **manchester** accepts two arguments - *len*, an integer representing the length of the list and *arr* and *arr*, a list of integers, respectively. Each element of *arr* represents a bit - 0 or 1

For example - if *arr* is {0 1 0 0 1 1 1 0}, the function/method should print an list {0 1 1 0 1 0 0 1}.

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Scores

The candidate did not make any changes in the code.

Compilation Statistics

0

Total attempts

0

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:00

Average time taken between two compile attempts:

00:00:00

Average test case pass percentage per compile:

0%

i 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.

i 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 5 (Language: Java)

The function/method **median** accepts two arguments - *size* and *inputList*, an integer representing the length of a list and a list of integers, respectively.

The function/method **median** is supposed to calculate and return an integer representing the median of elements in the input list. However, the function/method **median** works only for odd-length lists because of incomplete code.

You must complete the code to make it work for even-length lists as well. A couple of other functions/methods are available, which you are supposed to use inside the function/method **median** to complete the code.

Helper Description

The following class is used to represent a MedianCalculate and is already implemented in the default code (Do not write this definition again in your code):

```
class MedianCalculate
{
    public int[] items;

    public int first;

    public int last;

    public int order;

    int quick_select()
    {
        /*It calculate the median value

        This can be called as -

        MedianCalculate ob = new MedianCalculate(inputList, start_index, end_index, median_order);

        int res = ob.quick_select();

        where median_order is the half length of the inputList */
    }
}
```

Scores

The candidate did not make any changes in the code.

Compilation Statistics

0

Total attempts

0

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:00

Average time taken between two compile attempts:

00:00:00

Average test case pass percentage per compile:

0%

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 6 (Language: Java)

The function/method ***countOccurrence*** return an integer representing the count of occurrences of given value in the input list.

The function/method ***countOccurrence*** accepts three arguments - *len*, an integer representing the size of the input list, *value*, an integer representing the given value and *arr*, a list of integers, representing the input list.

The function/method ***countOccurrence*** compiles successfully but fails to return the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Scores

The candidate did not make any changes in the code.

Compilation Statistics

0

Total attempts

0

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:00

Average time taken between two compile attempts:

00:00:00

Average test case pass percentage per compile:

0%

i 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.

i 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 7 (Language: Java)

The function/method ***drawPrintPattern*** accepts *num*, an integer.

The function/method ***drawPrintPattern*** prints the first *num* lines of the pattern shown below.

For example, if *num* = 3, the pattern should be:

```
1 1
1 1 1 1
1 1 1 1 1 1
```

The function/method ***drawPrintPattern*** compiles successfully but fails to get the desired result for some test cases due to incorrect implementation of the function/method. Your task is to fix the code so that it passes all the test cases.

Scores

The candidate did not make any changes in the code.

Compilation Statistics

0

Total attempts

0

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:00

Average time taken between two compile attempts:

00:00:00

Average test case pass percentage per compile:

0%

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

4 | Learning Resources

English Comprehension			
Read novels to enhance your comprehension skills			
Read opinions to improve your comprehension			
Read research papers online			
Logical Ability			
Learn about Sherlock Holmes' puzzles and develop your deductive logic			
Take a course on advanced logic			
Play chess and challenge your reasoning skills			
Quantitative Ability (Advanced)			
Watch a video on the history of algebra and its applications			
Learn about proportions and its practical usage			
Learn about calculating percentages manually			
Icon Index			
Free Tutorial	Paid Tutorial	Youtube Video	Web Source
Wikipedia	Text Tutorial	Video Tutorial	Google Playstore