

Priyanshu Kumar

Test ID: 450016636260489 | 7979923120 | priyanshavkmr6@gmail.com

Test Date: February 24, 2026

Computer Science 100 /100	Logical Ability 62 /100	Computer Programming 80 /100	Quantitative Ability (Advanced) 69 /100
English Comprehension 83 /100	WriteX - Essay Writing 60 /100	Automata 95 /100	Automata Fix 58 /100
Personality Completed			

Computer Science			100 / 100
OS and Computer Architecture	DBMS	Computer Networks	
100 / 100	100 / 100	100 / 100	

Logical Ability			62 / 100
Inductive Reasoning	Deductive Reasoning	Abductive Reasoning	
61 / 100	63 / 100	62 / 100	

Computer Programming			80 / 100
Basic Programming	Data Structures	OOP and Complexity Theory	
80 / 100	80 / 100	79 / 100	

Quantitative Ability (Advanced)			69 / 100
Basic Mathematics	Advanced Mathematics	Applied Mathematics	
71 / 100	67 / 100	69 / 100	

English Comprehension

83 / 100

CEFR: **C2**

Grammar

Vocabulary

Comprehension

87 / 100

83 / 100

78 / 100

WriteX - Essay Writing

60 / 100

CEFR: **B1**

Content Score

Grammar Score

53 / 100

76 / 100

Automata

95 / 100

Programming Ability

Programming Practices

Functional Correctness

100 / 100

75 / 100

100 / 100

Automata Fix

58 / 100

Logical Error

Syntactical Error

Code Reuse

50 / 100

100 / 100

50 / 100

Personality

Completed

Competency	Score
Extraversion	66
Conscientiousness	56
Agreeableness	56
Openness to Experience	48
Emotional Stability	20
Polychronicity	2

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:

- $70 \leq \text{Score} < 100$
- $30 \leq \text{Score} < 70$
- $0 \leq \text{Score} < 30$

2 | Insights

English Comprehension

 83 / 100CEFR: **C2**

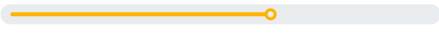
This test aims to measure your vocabulary, grammar and reading comprehension skills.

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

 62 / 100

Inductive Reasoning

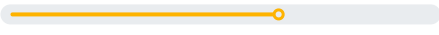
 61 / 100

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

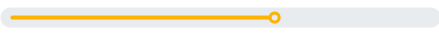
 63 / 100

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.



Abductive Reasoning

 62 / 100

Quantitative Ability (Advanced)

 69 / 100

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

You have a strong hold on basic arithmetic and concepts of algebra. You are able to convert real-world problems into equations and solve them.

Personality

Competencies



Extraversion

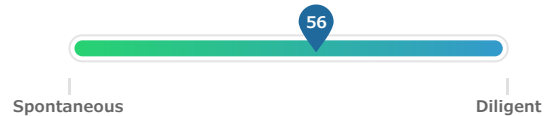


Extraversion refers to a person's inclination to prefer social interaction over spending time alone. Individuals with high levels of extraversion are perceived to be outgoing, warm and socially confident.

- 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 are flexible and able to adapt your work pace to the job at hand.
- You are usually spontaneous but you are likely to stick to a plan whenever necessary.
- You tend to be cautious when you deem it necessary.
- You may prefer to act according to the rules.
- You are confident in your ability to achieve goals but may need support to overcome occasional setbacks.
- You are an efficient worker and try to perform better than your peers. You are well suited for jobs allowing flexibility regarding operating procedures.



Agreeableness

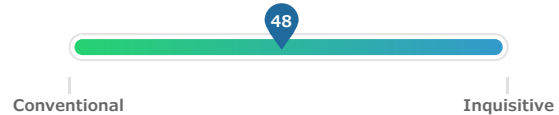


Agreeableness refers to an individual's tendency to be cooperative with others and it defines your approach to interpersonal relationships. People with high levels of this personality trait tend to be more considerate of people around them and are more likely to work effectively in a team.

- You are flexible regarding your opinions and be willing to accommodate the needs of others.
- You are generally considerate of the needs of others yet may, at times, overlook social norms to achieve personal success.
- You are selective about the people you choose to trust.
- You are caring and you empathise a friend in distress.
- You give credit to others but also tends to be open with your friends about personal achievements.
- You are more inclined to strike a compromise in tough situations and may be suitable for jobs that demand managing expectations among different stakeholders.



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 may try new things but would prefer not to venture too far beyond your comfort zone.
- You tend to be open to accepting abstract ideas after weighing them against existing solutions.
- You appreciate the arts to a certain extent but may lack the curiosity to explore them in depth.
- You may express your feelings only to people you are comfortable with.
- Your personality is more suited for jobs involving a mix of logical and creative thinking.



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 prefer to work on one task at a time, complete it and then move on to the next.
- You prefer orderliness and likes to concentrate on the task at hand without any distractions.
- You can find it difficult to be placed in a work environment where there is a need to multitask or where expected to engage in multiple projects simultaneously.

3 | Response

Automata

95 / 100
 [Code Replay](#)

Question 1 (Language: Java 11)

A company called Digicomparts manufactures 52 types of unique products for laptop and desktop computers. It manufactures 10 types of laptop products and 42 types of desktop products. Each product manufactured by the company has a unique productID from a-z and A-Z. The laptop products have productIDs (a, i, e, o, u, A, I, E, O, U) while the rest of the productIDs are assigned to the desktop products. The company manager wants to find the sales data for the desktop products.

Write an algorithm to help the manager find the productIDs of the desktop products.

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

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.

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 import java.util.*; 2 import java.lang.*; 3 import java.io.*; 4 5 /* 6 * productID representing the product IDs of the sales. 7 */ 8 public class Solution 9 { 10 public static int calculateDesktopProductIDs(char[] productID) 11 { 12 int answer = 0; 13 // Write your code here 14 String laptopIDs = "aeiouAEIOU"; 15 16 for(char c : productID){ 17 if(laptopIDs.indexOf(c) == -1) answer++; 18 } </pre>		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 Vulnerabilities and Errors Readability & Language Best Practices Line 16: Variables are given very short names.

```
19
20     return answer;
21 }
22
23 public static void main(String[] args)
24 {
25     Scanner in = new Scanner(System.in);
26     //input for productID
27     int productID_size = in.nextInt();
28     char productID[] = new char[productID_size];
29     for(int idx = 0; idx < productID_size; idx++)
30     {
31         productID[idx] = in.next().charAt(0);
32     }
33
34     int result = calculateDesktopProductIDs(productID);
35     System.out.print(result);
36
37 }
38 }
39
```

Test Case Execution

Passed TC: 100%



100%
Basic(10/10)

100%
Advance(4/4)

100%
Edge(1/1)

Compilation Statistics

1

Total attempts

1

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:	00:03:55
Average time taken between two compile attempts:	00:03:55
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 11)

An alternate sort of a list consists of alternate elements (starting from the first position) of the given list after sorting it in an ascending order. You are given a list of unsorted elements. Write an algorithm to find the alternate sort of the given list.

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



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.

Final Code Submitted

Compilation Status: Pass

```
1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
5 /*
6  * arr, representing the elements of input list.
7  */
8 public class Solution
```

Code Analysis

Average-case Time Complexity

Candidate code: $O(N \log N)$

Best case code: $O(N \log N)$

*N represents number of elements in the input list

```

9 {
10 public static int[] alternateSort(int[] arr)
11 {
12     // int[] answer = new int[100];
13     // Write your code here
14     Arrays.sort(arr);
15     int n = arr.length;
16     int resultSize = (n+1)/2;
17     int[] answer = new int[resultSize];
18     int count = 0;
19     for(int i = 0; i < n; i+=2){
20         answer[count++] = arr[i];
21     }
22     return answer;
23 }
24
25 public static void main(String[] args)
26 {
27     Scanner in = new Scanner(System.in);
28     //input for arr
29     int arr_size = in.nextInt();
30     int arr[] = new int[arr_size];
31     for(int idx = 0; idx < arr_size; idx++)
32     {
33         arr[idx] = in.nextInt();
34     }
35
36     int[] result = alternateSort(arr);
37     for(int idx = 0; idx < result.length - 1; idx++)
38     {
39         System.out.print(result[idx] + " ");
40     }
41     System.out.print(result[result.length - 1]);
42 }
43 }
44

```

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilities and Errors

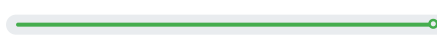
Readability & Language Best Practices

Line 15: Variables are given very short names.

Test Case Execution

Passed TC: 100%

Total score

 24/24

100%

Basic(11/11)

100%

Advance(11/11)

100%

Edge(2/2)

Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

0

Sample failed

0

Timed out

1

Runtime errors

Response time:

00:06:11

Average time taken between two compile attempts:

00:03:06

Average test case pass percentage per compile:

50%

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

Automata Fix



58 / 100

[Code Replay](#)

Question 1 (Language: Java)

The function/method ***removeElement*** prints space separated integers that remains after removing the integer at the given index from the input list.

The function/method ***removeElement*** accepts three arguments - *size*, an integer representing the size of the input list, *indexValue*, an integer representing given index and *inputList*, a list of integers representing the input list.

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

Note:

Zero-based indexing is followed to access list elements.

Scores

Final Code Submitted

Compilation Status: Pass

```

1 // You can print the values to stdout for debugging
2 class Solution
3 {
4     void removeElement(int size, int indexValue, int[] inputList)
5     {
6         int i;
7         if(indexValue >= 0 && indexValue < size )
8         {
9             for( i = indexValue ; i < size - 1 ; i ++ )
10            {
11                inputList[i] = inputList[i+1];
12            }
13            for(i=0;i<size-1;i++)
14                System.out.print(inputList[i]+" ");
15        }
16        else
17        {
18            for(i=0;i<size;i++)
19                System.out.print(inputList[i]+" ");
20        }
21    }
22 }
```

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:

*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: 100%

Total score



100%

Basic(5/5)

100%

Advance(2/2)

100%

Edge(1/1)

Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

1

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:04:04

Average time taken between two compile attempts:

00:02:02

Average test case pass percentage per compile:

56.3%

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 2 (Language: Java)

The function/method **countElement** returns the number of elements in the input list `arr` which are greater than twice the input number `K`. The function/method **countElement** accepts three arguments - `size`, an integer representing the size of the input list, `numK`, an integer representing the input number `K` and `inputList`, a list of integers.

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

Scores

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 // You can print the values to stdout for debugging 2 class Solution 3 { 4 int countElement(int size, int numK, int[] inputList) 5 { 6 int count=0; 7 for(int i=0;i<size;i++) 8 { 9 if(inputList[i]>(2 * numK)) 10 count+=1; 11 } 12 return count; 13 } 14 } 15 16 </pre>		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:
		*N represents
		Errors/Warnings
		There are no errors in the candidate's code.
		Structural Vulnerabilities and Errors
		There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score

10/10

100%

Basic(6/6)

100%

Advance(3/3)

100%

Edge(1/1)

Compilation Statistics

5

Total attempts

1

Successful

4

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:04:34

Average time taken between two compile attempts:

00:00:55

Average test case pass percentage per compile:

100%

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 3 (Language: Java)

The function/method **countDigits** return an integer representing the remainder when the given number is divided by the number of digits in it.

The function/method **countDigits** accepts an argument - *num*, an integer representing the given number.

The function/method **countDigits** 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

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 // You can print the values to stdout for debugging 2 class Solution 3 { 4 public int countDigits(int num) 5 { 6 int count = 0; 7 int temp = num; 8 9 if(num == 0) return 0; 10 while(temp != 0){ 11 temp = temp / 10; 12 count ++; 13 } 14 return (num % count); 15 } 16 }</pre>		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: *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: 100%		
Total score 8/8	100% Basic(6/6)	100% Advance(2/2)	0% Edge(0/0)

Compilation Statistics					
1 Total attempts	1 Successful	0 Compilation errors	0 Sample failed	0 Timed out	0 Runtime errors
Response time:					00:02:51
Average time taken between two compile attempts:					00:02:51
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 4 (Language: Java)

You are given predefined structure **Time** containing *hour*, *minute*, and *second* as members. A collection of functions/methods for performing some common operations on times is also available. You must make use of these functions/methods to calculate and return the difference.

The function/method ***difference_in_times*** accepts two arguments - *time1*, and *time2*, representing two times and is supposed to return an integer representing the difference in the number of seconds.

You must complete the code so that it passes all the test cases.

.

Helper Description

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

```
public class Time
{
    public int hour;
    public int minute;
    public int second;
    public Time(int h, int m, int s)
    {
        hour = h;
```

```

minute = m;

second = s;

}

public int compareTo(Object anotherTime)
{
    /*Return 1, if time1 is greater than time2.

    Return -1 if time1 is less than time2

    or, Return 0, if time1 is equal to time2

    This can be called as -

    * If time1 and time2 are two Time then -

    * time1.compareTo(time2) */
}

public void addSecond()
{
    /* Add one second in the time;

    This can be called as -

    * If time1 is Time then -

    * time1.addSecond() */
}

```

Scores

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 // You can print the values to stdout for debugging 2 class Solution 3 { 4 int difference_in_times(Time time1, Time time2) 5 { 6 // write your code here 7 int t1seconds = (time1.hour * 3600) + (time1.minute * 60) + (time1.second); 8 9 int t2seconds = (time2.hour * 3600) + (time2.minute * 60) + (time2.second); 10 11 return t1seconds - t2seconds; 12 } 13 } </pre>		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: *N represents
		Errors/Warnings There are no errors in the candidate's code.
		Structural Vulnerabilities and Errors

14

There are no errors in the candidate's code.

Test Case Execution

Passed TC: 55.56%

Total score



80%

Basic(4/5)

0%

Advance(0/3)

100%

Edge(1/1)

Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

2

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:04:47

Average time taken between two compile attempts:

00:02:24

Average test case pass percentage per compile:

11.1%

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 **findMaxElement** return an integer representing the largest element in the given two input lists.

The function/method **findMaxElement** accepts four arguments - *len1*, an integer representing the length of the first list, *arr1*, a list of integers representing the first input list, *len2*, an integer representing the length of the second input list and *arr2*, a list of integers representing the second input list, respectively.

Another function/method **sortArray** accepts two arguments - *len*, an integer representing the length of the list and *arr*, a list of integers, respectively and return a list sorted ascending order.

Your task is to use the function/method **sortArray** to complete the code in **findMaxElement** so that it passes all the test cases.

Scores

Final Code Submitted

Compilation Status: Pass

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:

*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.

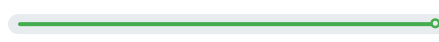
```

1 // You can print the values to stdout for debugging
2 class Solution
3 {
4     public int[] sortArray(int len, int[] arr)
5     {
6         int i=0,j=0,temp=0;
7         for(i=0;i<arr.length;i++)
8         {
9             for(j=i+1;j<arr.length;j++)
10            {
11                if(arr[i]>arr[j])
12                {
13                    temp = arr[i];
14                    arr[i] = arr[j];
15                    arr[j] = temp;
16                }
17            }
18        }
19        return arr;
20    }
21
22    public int findMaxElement(int len1, int[] arr1, int len2, int[] arr2)
23    {
24        // write your code here and return maximum element
25        arr1 = sortArray(len1, arr1);
26        arr2 = sortArray(len2, arr2);
27
28        int max1 = arr1[len1-1];
29        int max2 = arr2[len2-1];
30
31        if(max1> max2){
32            return max1;
33        }else{
34            return max2;
35        }
36    }
37 }
```

Test Case Execution

Passed TC: 100%

Total score

 8/8

100%

Basic(2/2)

100%

Advance(6/6)

0%

Edge(0/0)

Compilation Statistics

1

Total attempts

1

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:02:03

Average time taken between two compile attempts:

00:02:03

Average test case pass percentage per compile:

100%

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 6 (Language: Java)

The function/method **arrayReverse** modify the input list by reversing its element

The function/method **arrayReverse** accepts two arguments - *len*, an integer representing the length of the list and *arr*, list of integers representing the input list, respectively.

For example, if the input list *arr* is {20 30 10 40 50}, the function/method is supposed to print {50 40 10 30 20}.

The function/method **arrayReverse** compiles successfully but fails to get 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

Final Code Submitted

Compilation Status: Fail

```
1 // You can print the values to stdout for debugging
2 class Solution
3 {
4     public void arrayReverse(int len, int[] arr)
5     {
6         int i, temp, originalLen = len;
7         for( i = 0 ; i < originalLen / 2 ; i++ )
8         {
9             temp = arr[i];
10            arr[i] = arr[originalLen - 1-i]
11            len -= 1;
12        }
13    }
14 }
```

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:

*N represents

Errors/Warnings

Solution.java:10: error: ';' expected
arr[i] = arr[originalLen - 1-i]
^
1 error

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

0

Total attempts

0

Successful

0

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:01:19

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 7 (Language: Java)

The function/method **printCharacterPattern** accepts an integer *num*. It is supposed to print the first *num* ($0 \leq num \leq 26$) lines of the pattern as shown below.

For example, if *num* = 4, the pattern is:

```
a
ab
abc
abcd
```

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

WriteX - Essay Writing



60 / 100

CEFR: B1

Question

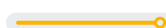
These days, children are becoming brash and rough at an early age because of their exposure to television and internet.

Do you agree or disagree? Support your stand with reasons.

Scores

Content Score

Grammar Score



53 / 100



76 / 100

Response

In the contemporary digital age, children are "digital natives," often engaging with screens as early as four months old. While these platforms offer educational benefits, unfettered access to television and the internet has been increasingly linked to impulsive, "brash," and physically "rough" behaviors in early childhood. Observational Lea

Error Summary



Spelling

0

ning and **Mimicry**: According to social learning theory, children have an innate tendency to mimic what they observe. When exposed to "charismatic heroes" or characters who use aggression to solve problems or achieve goals, children often adopt these "scripts" for their own real-world interactions. Desensitization to Violence: Repeated exposure to violent or aggressive media can "numb" a child's natural emotional response to the suffering of others. This desensitization makes it easier for them to plan or engage in aggressive acts without feeling the typical negative emotions, such as guilt or anxiety, that would otherwise inhibit such behavior. Instant Gratification and **Impulsivity**: The fast-paced nature of internet contentsuch as TikTok videos or Instagram reelscan reduce attention spans and **condition** children for instant gratification. This often leads to increased impulsivity, which manifests as "brash" or rude behavior when their immediate desires are not met. Reduced Prosocial **Interaction**: Excessive screen time often replaces vital face-to-face social interactions with peers and **family**. Without these real-world experiences, children may fail to develop essential social skills, such as empathy, conflict resolution, and self-regulation, leading to rougher social conduct. Studies have shown that children who watch many hours of violent television in elementary school are more likely to exhibit aggressive behavior as teenagers and even as adults. For example, one longitudinal study found that children in the highest quartile of violence viewing were significantly more likely to have committed crimes or engaged in physical altercations by age 26 compared to their peers. While **media** is not the sole cause of behavioral issues, its role as a significant environmental factor is well-documented. The combination of modeling aggressive behavior, desensitizing young minds to harm, and fostering an impulsive mindset creates a framework where children are more likely to behave in a brash and rough manner.

	White Space	3
	Style	0
	Grammar	7
	Typographical	14

Essay Statistics

343 Total words	14 Total sentences	25 Average sentence length	231 Total unique words	116 Total stop words
---------------------------	------------------------------	--------------------------------------	----------------------------------	--------------------------------

Error Details

White Space

...y "rough" behaviors in early childhood. Observational Learning and Mimicry: Acco...	Possible typo: you repeated a whitespace
...ion, leading to rougher social conduct. Studies have shown that children who wat...	Possible typo: you repeated a whitespace
...ions by age 26 compared to their peers. While media is not the sole cause of beh...	Possible typo: you repeated a whitespace

Grammar

Observational Learning and Mimicry : According to social learning theory, children have an innate tendency to mimic what they observe.	Possible grammar error found. Consider replacing it with "Mimicry,".
---	--

Desensitization to Violence: Repeated exposure to violent or aggressive media can "numb" a child's natural emotional response to the suffering of others. This desensitization makes it easier for them to plan or engage in aggressive acts without feeling the typical negative emotions, such as guilt or anxiety, that would otherwise inhibit such behavior.

Possible grammar error found. Consider inserting "is" over here.

Instant Gratification and Impulsivity: The fast-paced nature of internet contentsuch as TikTok videos or Instagram reelscan reduce attention spans and condition children for instant gratification.

Possible grammar error found. Consider replacing it with "Impulsivity,".

Instant Gratification and Impulsivity: The fast-paced nature of internet contentsuch as TikTok videos or Instagram reelscan reduce attention spans and condition children for instant gratification.

Possible grammar error found. Consider replacing it with "conditions".

Reduced Prosocial Interaction: Excessive screen time often replaces vital face-to-face social interactions with peers and family.

Possible grammar error found. Consider replacing it with "Interactions".

Reduced Prosocial Interaction: Excessive screen time often replaces vital face-to-face social interactions with peers and family.

Possible grammar error found. Consider replacing it with "family members.".

For example, one longitudinal study found that children in the highest quartile of violence viewing were significantly more likely to have committed crimes or engaged in physical altercations by age 26 compared to their peers. While media is not the sole cause of behavioral issues, its role as a significant environmental factor is well-documented.

Possible grammar error found. Consider inserting "the" over here.

Typographical

... contemporary digital age, children are "digital natives," often engaging with screens...

Use a smart opening quote here: ""

...ital age, children are "digital natives," often engaging with screens as early as...

Use a smart closing quote here: ""

... been increasingly linked to impulsive, "brash," and physically "rough" behaviors...

Use a smart opening quote here: ""

...ncreasingly linked to impulsive, "brash," and physically "rough" behaviors in early childhood. Observa...

Use a smart closing quote here: ""

...d to impulsive, "brash," and physically "rough" behaviors in early childhood. Observa...

Use a smart opening quote here: ""

...mpulsive, "brash," and physically "rough" behaviors in early childhood. Observa...

Use a smart closing quote here: ""

...imic what they observe. When exposed to "charismatic heroes" or characters who use...

Use a smart opening quote here: ""

...rve. When exposed to "charismatic heroes" or characters who use aggression to solve...

Use a smart closing quote here: ""

...hieve goals, children often adopt these "scripts" for their own real-world interactions...

Use a smart opening quote here: ""

...als, children often adopt these "scripts" for their own real-world interactions. ...

Use a smart closing quote here: ""

...sure to violent or aggressive media can "numb" a child's natural emotional respon...

Use a smart opening quote here: ""

...to violent or aggressive media can "numb" a child's natural emotional response to...

Use a smart closing quote here: ""

...creased impulsivity, which manifests as "brash" or rude behavior when their immedi...

Use a smart opening quote here: ""

...d impulsivity, which manifests as "brash" or rude behavior when their immediate d...

Use a smart closing quote here: ""

4 | Learning Resources

English Comprehension

[Read novels to enhance your comprehension skills](#)



[Read opinions to improve your comprehension](#)



[Read research papers online](#)



Logical Ability

[Take a course on advanced logic](#)



[Play chess and challenge your reasoning skills](#)



[Practice inductive reasoning on visual data](#)



Quantitative Ability (Advanced)

[Learn about basic math for adults](#)



[Watch a video on how to calculate your Income Tax](#)



[Watch a video on probability](#)



Icon Index



Free Tutorial



Paid Tutorial



Youtube Video



Web Source



Wikipedia



Text Tutorial



Video Tutorial



Google Playstore