Jashwanth Ram K

Test ID: 413720353809974 | 9884886529 | jash22052.cs@rmkec.ac.in

Test Date: July 19, 2025

Computer Science	67 /100
Logical Ability	41 /100
Computer Programming	61 /100
Quantitative Ability (Advanced)	68 /100
English Comprehension	54 /100
WriteX - Email Writing	62 /100
Automata	85 /100
Automata Fix	58 /100
Personality	✓ Completed

Computer Science

67 / 100

OS and Computer Architecture	71 / 100
DBMS	45 / 100
Computer Networks	84 / 100

Logical Ability

41 / 100

Inductive Reasoning	41 / 100
Deductive Reasoning	43 / 100
Abductive Reasoning	39 / 100

Computer Programming

61 / 100

Basic Programming	Data Structures	OOP and Complexity Theory
57 / 100	65 / 100	62 / 100

Quantitative Ability (Advanced)

68 / 100

Basic Mathematics	Advanced Mathematics	Applied Mathematics
67 / 100	65 / 100	73 / 100

English Comprehension

54 / 100 | CEFR: **B2**

Grammar	Vocabulary	Comprehension
53 / 100	57 / 100	51 / 100

WriteX - Email Writing

62 / 100 | CEFR: **A2**

Etiquette	Content	Grammar
83 / 100	50 / 100	54 / 100

Automata

85 / 100

Programming Ability	Programming Practices	Functional Correctness
100 / 100	25 / 100	100 / 100

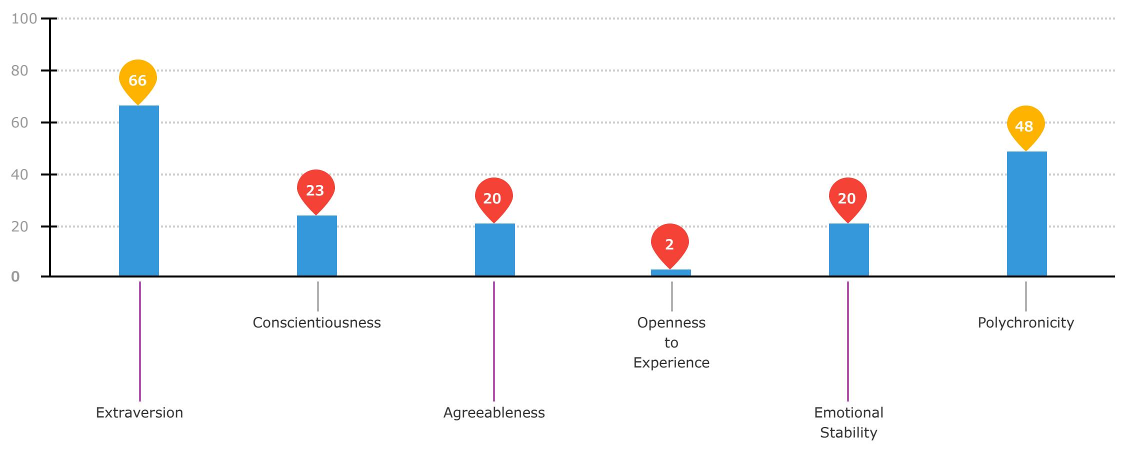
Automata Fix

58 / 100

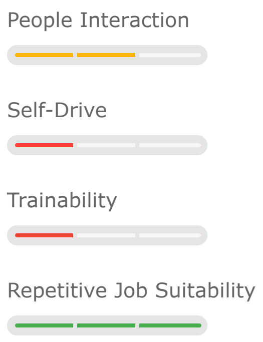
Logical Error	Code Reuse	Syntactical Error
75 / 100	50 / 100	0 / 100

Personality

Completed



Competencies



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

54 / 100 | CEFR: **B2**

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

You are able to construct short sentences and understand simple text. The ability to read and comprehend is important for most jobs. However, it is of utmost importance for jobs that involve research, content development, editing, teaching, etc.

Logical Ability

41 / 100

Inductive Reasoning

41 / 100

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

You are able to work out simple rules based on specific evidence or information. This skill is required in high end analytics jobs where one is required to infer patterns based on predefined rules from different sets of data.

Deductive Reasoning

43 / 100

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

You are able to work out simple rules based on specific evidence or information. This skill is required in high end analytics jobs where one is required to infer patterns based on predefined rules from different sets of data.

Abductive Reasoning

39 / 100

Quantitative Ability (Advanced)

68 / 100

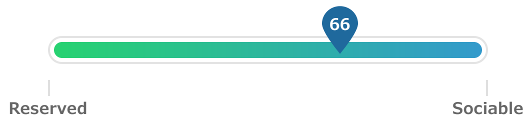
This test aims to measure your ability to solve problems on basic arithmetic operations, probability, permutations and combinations, and other advanced concepts.

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

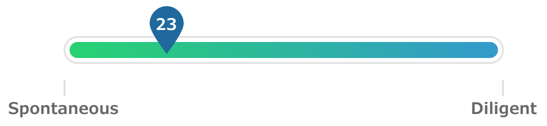
Extraversion



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.

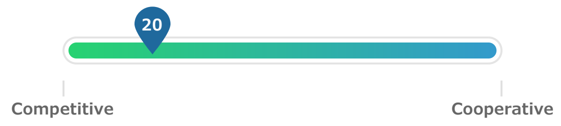
Conscientiousness



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 have an easy going attitude towards work.
- You tend to act spontaneously and may prefer to get started on a project or at work without laying out an organized plan.
- You tend to make quick decisions rather than spending time on reviewing the facts at hand.
- You have a flexible and spontaneous approach to work and is less likely to adhere to strict guidelines and policies.
- You sometimes feel insecure about your capabilities and could benefit from emotional support.
- You are likely to be content with your level of achievement and may not strive to achieve more.
- Your personality is more suited for jobs which demand fast results and are not as concerned with attention to detail.

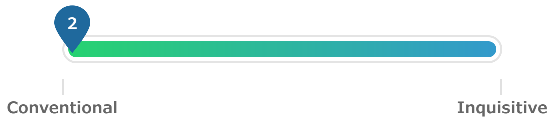
Agreeableness



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 outspoken. You often play the role of a devil's advocate in discussions and question others' opinions and views.
- You are not gullible and are likely to carefully examine the situation before trusting something/someone.
- You may not be strongly affected by human suffering and may be perceived as indifferent.
- You are confident of your achievements and do not shy away from talking about them.
- You sometimes place self-interest above the needs of those around you. You are not willing to compromise your own views in order to accommodate the views of others.
- You are suitable for jobs that require tough objective decisions and hard negotiation.

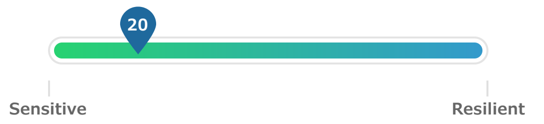
Openness to Experience



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 not be very open to new experiences lying outside your comfort zone and tends to prefer routine over variety.
- You may be pragmatic and is likely to be conventional in your outlook and actions and may not pursue an experimental approach to problem-solving.
- You may not have an appreciation for art.
- You do not like to express your emotions and feelings to others.
- You tend to demonstrate concrete thinking with a focus on practical solutions, as opposed to abstract ideas.
- Your personality is more suited to job roles that require logical and rational thinking.

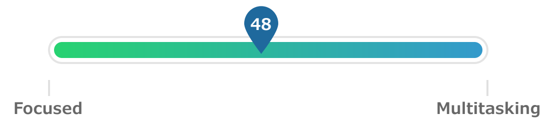
Emotional Stability



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 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.

WriteX - Email Writing

62 / 100 | CEFR: **A2**

Etiquette

83 / 100

This score aims to measure the ability to write an email in a proper format. A well written email is easier to read and understand. This skill is extremely important at work given the extensive use of emails for official communication.

Your email is almost free of etiquette errors. You generally follows the best practices of writing an email.

Suggestion to Improve

- You should ensure that you doesn't overuse the high priority option, otherwise few people will take it seriously.
- You should make sure whether your tone and intent are clear in the email.

- You should be careful not to overuse punctuation, but at the same time not be afraid to add personality and emotions to your emails. An email that portrays you as a friendly, yet serious business person will create a favorable impression.

Content

50 / 100

This score measures the relevance of the content written on a given topic. A high score on this competency means that the content is relevant, simple and descriptive. The reasoning provided in the email has the ability to influence the reader. This competency is important in most jobs as written communication with clients and colleagues is an integral part of any job.

- The content is comprehensible and the main subject has some relevant points supporting it, but lacks certain minor details. The email has room for improvement.

Suggestion to Improve

- You should use headings, subheadings, bullet points, and numbering whenever possible to break up the text.
- You should also remember to put all the required details. The reader might not be familiar with the topic and could misunderstand an important point.
- You should always proofread your email before hitting the "send" button.

Grammar

54 / 100

This score aims to measure the grammatical correctness of the email written by you. Improper grammar can affect the meaning and clarity of the email. Knowledge of basic sentence structure and avoiding grammatical errors ensures effective communication.

- Your sentences are properly structured (without any fragmented sentence) but there are noticeable grammatical errors.

Suggestion to Improve

- You should write the information in an email clearly and concisely.
- You should make sure to do a spell check and a grammar check with the help of free tools available online. You should avoid the use of abbreviation in the email.
- You should always proofread your email before hitting the "send" button.

3 | Response

Automata ● 85 / 100 | Code Replay

Question 1 (Language: Java) High similarity detected : **95%**

A company manufactures different types of software products. They deliver their products to their N clients. Whenever the company fulfils the complete order of a client the orderID generated is the sum of the number of products delivered for every committed product type. The head of the sales team wants to find the client-specific data for the total number of products of any type delivered to each client.

Write an algorithm for the head of the sales team to calculate the total number of products of any type delivered to the respective clients.

Scores

Programming Ability

100 / 100

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

Programming Practices

25 / 100

Low readability, low on program structure. The source code is poorly formatted and contains redundant/improper coding constructs.

Functional Correctness

100 / 100

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

Final Code Submitted **Compilation Status: Pass**

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

/*
 *
 */
public class Solution
{
    public static int[]  prodDelivery(int[] orderId)
    {
        int[]  answer = new int[orderId.length];
        // Write your code here
        for(int i=0;i<orderId.length;i++)
        {
```

Code Analysis

Average-case Time Complexity

Candidate code: O(N)

Best case code: O(N)

*N represents number of clients.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

```
16            answer[i] = func(orderId[i]);
17        }
18        return answer;
19
20
21
22
23    }
24
25    public static int func(int num)
26    {
27        int x =num;
28        int sum = 0;
29        while(num>0)
30        {
31            int d = num%10;
32            sum+=d;
33            num/=10;
34        }
35        return sum;
36    }
37
38    public static void main(String[] args)
39    {
40        Scanner in = new Scanner(System.in);
41        //input for orderID
42        int orderID_size = in.nextInt();
43        int orderID[] = new int[orderID_size];
44        for(int idx = 0; idx < orderID_size; idx++)
45        {
46            orderID[idx] = in.nextInt();
47        }
48
49        int[] result = prodDelivery(orderID);
50        for(int idx = 0; idx < result.length - 1; idx++)
51        {
52            System.out.print(result[idx] + " ");
53        }
54        System.out.print(result[result.length - 1]);
55    }
56 }
57
```

Readability & Language Best Practices

Line 27,31: Variables are given very short names.

Performance & Correctness

Line 27: Premature Declaration of variables must be avoided.
Line 27: Local variable is declared and/or assigned, but not used.

Test Case Execution

Passed TC: **100%**

Total score 16/16

100% Basic(**7**/7)

100% Advance(**7**/7)

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

Compilation Statistics

7	4	3	0	1	1
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:08:10**
Average time taken between two compile attempts:	**00:01:10**
Average test case pass percentage per compile:	**28.57%**

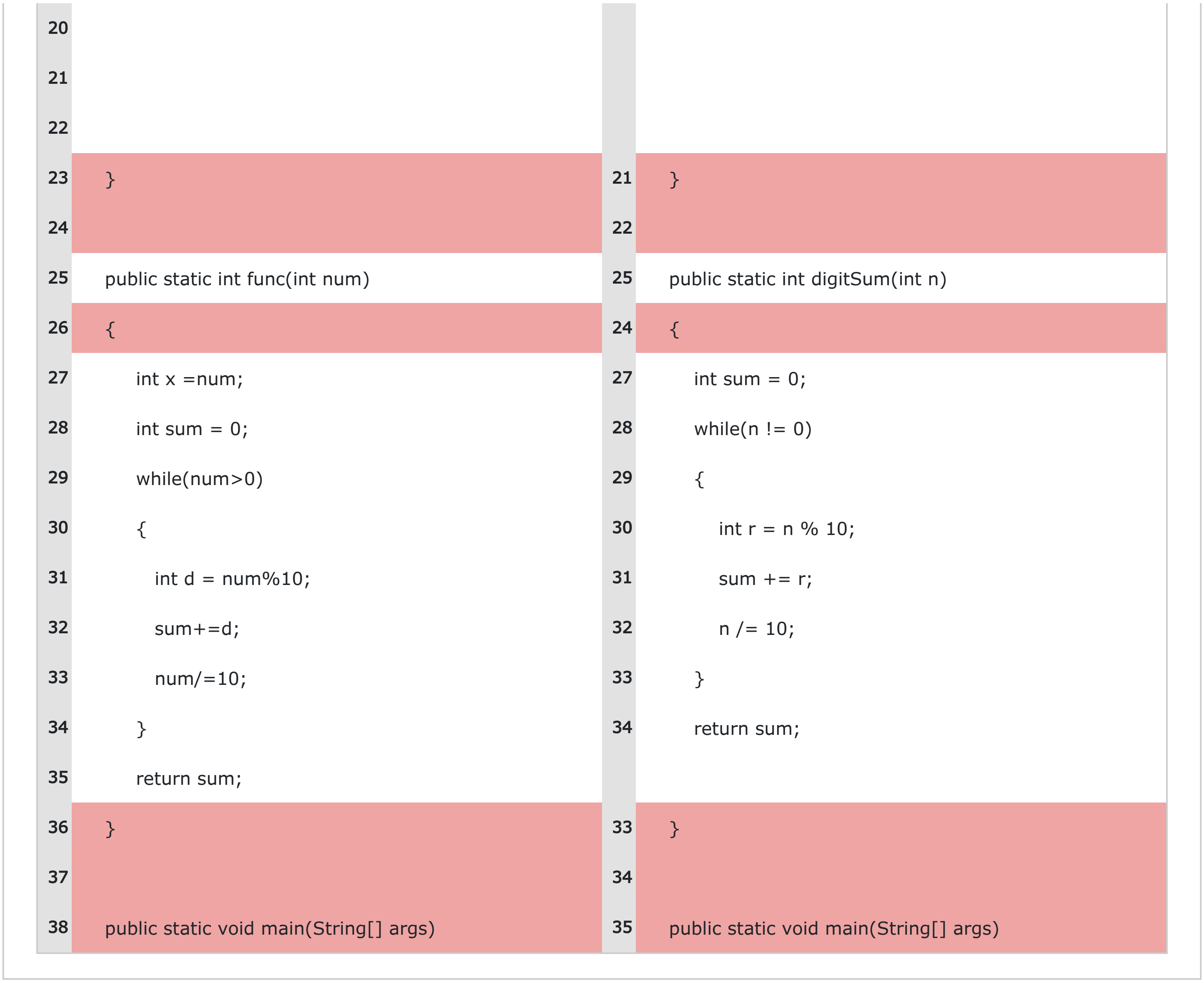
Similarity Detected

Every response is checked against:

- The responses of peers who took this assessment in the same event
- All candidate responses submitted in the last week
- Content currently available on the web

This response has a high degree of similarity to one of these sources, which means it is possibly not the candidate’s original work:

Candidate Response	Peer's Response
`*/`	`*/`
`public class Solution`	`public class Solution`
`{`	`{`
`public static int[] prodDelivery(int[] orderId)`	`public static int[] prodDelivery(int[] orderID)`
`{`	`{`
`int[] answer = new int[orderId.length];`	`int[] answer = new int[orderID.length];`
`// Write your code here`	`// Write your code here`
`for(int i=0;i<orderId.length;i++)`	
	`for(int i = 0; i < orderID.length; i++)`
`{`	`{`
`answer[i] = func(orderId[i]);`	`answer[i] = digitSum(orderID[i]);`
`}`	`}`
`return answer;`	`return answer;`



```
}

public static int func(int num)
{
    int x =num;
    int sum = 0;
    while(num>0)
    {
      int d = num%10;
      sum+=d;
      num/=10;
    }
    return sum;
}

public static void main(String[] args)
```

```
}

public static int digitSum(int n)
{
    int sum = 0;
    while(n != 0)
    {
        int r = n % 10;
        sum += r;
        n /= 10;
    }
    return sum;
}

public static void main(String[] args)
```

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)

In a town, the houses are marked with letters in the English alphabet. A town committee wants to renovate each house. Because funds are limited, they decide to renovate only the houses marked with vowels. The committee head gives the list of houses to the members and asks them to identify the houses that will not be renovated.

Write an algorithm to help the committee members find the list of houses that will not be renovated.

Scores

Programming Ability

100 / 100

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

Programming Practices

25 / 100

Low readability, low on program structure. The source code is poorly formatted and contains redundant/improper coding constructs.

Functional Correctness

100 / 100

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

Final Code Submitted   Compilation Status: Pass

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

Code Analysis

Average-case Time Complexity

```
 4
 5 /*
 6  *
 7  */
 8 public class Solution
 9 {
10    public static String  renovateHouses(String houses)
11    {
12       String  answer = "";
13       for(char x:houses.toCharArray())
14       {
15          if(!vow(x))
16          {
17             answer+=x;
18          }
19       }
20
21
22
23
24       return answer;
25    }
26
27    public static boolean vow(char x)
28    {
29       if(x=='a'||x=='e'||x=='i'||x=='o'||x=='u'||x=='A'||x=='E'||x=
   ='I'||x=='O'||x=='U')
30       {
31          return true;
32       }
33       return false;
34    }
35
36    public static void main(String[] args)
37    {
38       Scanner in = new Scanner(System.in);
39
40       // input for houses
41       String houses = in.nextLine();
42
43       String result = renovateHouses(houses);
44       System.out.print(result);
45
46    }
47 }
48
```

Candidate code: O(N)

Best case code: O(N)

*N represents number of houses

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

Readability & Language Best Practices

Line 13,27: Variables are given very short names.
Line 27: The method 'vow' has a Cyclomatic Complexity of 11.

Performance & Correctness

Line 17: Prefer StringBuffer over += for concatenating strings

Test Case Execution

Passed TC: **100%**

Total score 20/20

100%	**100%**	**100%**
Basic(**6**/6)	Advance(**12**/12)	Edge(**2**/2)

Compilation Statistics

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

Response time:	**00:07:27**
Average time taken between two compile attempts:	**00:03:44**
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

Automata Fix

58 / 100 Code Replay

Question 1 (Language: Java)

The function/method ***productMatrix*** accepts three arguments - *rows,* an integer representing the rows of the matrix; *columns,* an integer representing the columns of the matrix and *matrix*, a two-dimensional array of integers, respectively.

The function/method ***productMatrix*** return an integer representing the product of the odd elements whose ith and jth

The function/method ***productMatrix*** return an integer representing the product of the odd elements whose i th and j
index are the same. Otherwise, it returns 0.

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

Scores

Final Code Submitted **Compilation Status: Pass**

```
public class Solution
{
  public int productMatrix(int rows, int columns, int matrix[][])
   {
      int result=1;
      boolean dd =false;
      for(int i=0;i<rows;i++)
      {
         for(int j=0;j<columns;j++)
         {
            if((i==j) && (matrix[i][j]%2!=0))
            {
               result *=matrix[i][j];
               dd =true;
            }
         }
   }

   if(dd)
   {
      return result;
   }

         return 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.

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(**5**/5)

100% Advance(**2**/2)

100% Edge(**1**/1)

Compilation Statistics

5	5	0	2	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:03:23**
Average time taken between two compile attempts:	**00:00:41**
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 ***replaceValues*** is modifying the input list in such a way - if the length of input list is odd, then all the elements of the input list are supposed to be replaced by 1s and in case it is even, the elements should be replaced by 0s.

For example: given the input list [0 1 2], the function will modify the input list like [1 1 1]

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

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

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
   void  replaceValues(int size, int[] inputList)
   {
      int i , j;
      if( size % 2 == 0 )
      {
         i=0;
         while(i<size)
         {
            inputList[i] = 0;
            i+=1;
         }
      }
      else
      {
         j=0;
         while(j<size)
         {
            inputList[j] = 1;
            j+=1;
         }
      }
   }
}

```

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 8/8

100%	**0%**	**100%**
Basic(**6**/6)	Advance(**0**/0)	Edge(**2**/2)

Compilation Statistics

2	2	0	1	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:01:20**
Average time taken between two compile attempts:	**00:00:40**
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 3 (Language: Java)

The function/method ***calculateMatrixSum*** returns an integer representing the sum of odd elements of the given matrix whose ith and jth index are the same.

The function/method ***calculateMatrixSum*** accepts three arguments - *rows*, an integer representing the number of rows of the given matrix, *columns*, an integer representing the number of columns of the given matrix and *matrix*, representing a two-dimensional array of integers.

The function/method ***calculateMatrixSum*** 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

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdou
t for debugging
class Solution
{
  int calculateMatrixSum(int rows, int columns, int matrix[][])
  {
    int i, j, sum=0;

      for(i=0;i<rows;i++)
      {
        for(j=0;j<columns;j++)
        {
          if(i==j)
          {
            if(matrix[i][j]%2!=0)
              sum += matrix[i][j];
          }
        }
      }
      return sum;

  }
}
```

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 8/8	**100%** Basic(**4**/4)	**100%** Advance(**3**/3)	**100%** Edge(**1**/1)

Compilation Statistics

6	6	0	3	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:01:53**
Average time taken between two compile attempts:	**00:00:19**
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 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**

```
// You can print the values to stdout for debugging
class Solution
{

  public int func(Time time1)
  {
    int h = time1.hour;
    int min = time1.minute;
    int sec = time1.second;
```

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

```

        return h*3600+min*60+sec;
    }
    int difference_in_times(Time time1, Time time2)
    {
      int val1 = func(time1);
      int val2 =func(time2);
      return Math.abs(val1-val2);
    }
}

```

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 9/9

100%	**100%**	**100%**
Basic(**5**/5)	Advance(**3**/3)	Edge(**1**/1)

Compilation Statistics

3	1	2	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:04:47**
Average time taken between two compile attempts:	**00:01:36**
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 5 (Language: Java)

The function/method ***allExponent*** returns a real number representing the result of exponentiation of base raised to power exponent for all input values. The function/method ***allExponent*** accepts two arguments - *baseValue*, an integer representing the base and *exponentValue*, an integer representing the exponent.

The incomplete code in the function/method ***allExponent*** works only for positive values of the exponent. You must complete the code and make it work for negative values of exponent as well.

Another function/method ***positiveExponent*** uses an efficient way for exponentiation but accepts only positive *exponent* values. You are supposed to use this function/method to complete the code in ***allExponent*** function/method.

Helper Description

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

```
public class Exponent

{

   public int base;

   public int exponent;

   int positiveExponent()

   {

       /*It calculate the Exponent for positive value of exponentValue

        This can be called as -

        Exponent exp = new Exponent(baseValue, exponentValue);

        float res = exp.positiveExponent();*/

   }

}
```

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
   float allExponent(int baseValue, int exponentValue)
   {
```

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.

```
    float res = 1;
      return res;
  }
}
```

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: **16.67%**

Total score

1/6

0%	0%	100%
Basic(**0**/2)	Advance(**0**/3)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:02:18**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**16.7%**

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 ***sortArray*** modify the input list by sorting its elements in descending order.
The function/method ***sortArray*** accepts two arguments - *len,* representing the length of the list and *arr,* a list of integers representing the input list, respectively*.*

The function/method ***sortArray*** 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 sortArray(int len, int[] arr)
5    {
6
7      Arrays.sort(arr);
8      int[] res =arr;
9      int it = 0;
10     for(int i=res.length-1;i>=0;i--)
11     {
12       arr[it++] = res[i];
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:7: error: cannot find symbol
```

```
  }
}
```

```
Arrays.sort(arr);
^
symbol: variable Arrays
location: class Solution
1 error
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

4	1	3	1	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:02:59**
Average time taken between two compile attempts:	**00:00:45**
Average test case pass percentage per compile:	**12.5%**

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 ***maxReplace*** prints space separated integers representing the input list after replacing all elements of the input list with the maximum element of the input list.

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

The function/method ***maxReplace*** compiles unsuccessfully due to compilation error. Your task is to debug 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    void maxReplace(int size, int[] inputList)
 5    {
 6
 7          int max =inputList[0];
 8          for(int i=1;i<size;i++)
 9          {
10             if(max<inputList[i])
11             {
12                 max = inputList[i];
13             }
14          }
15
16       for(int i=0;i<size,i++)
17       {
18          inputList[i]=max
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

```
Solution.java:16: error: ';' expected
for(int i=0;i ^
Solution.java:16: error: ')' expected
for(int i=0;i ^
Solution.java:16: error: illegal start of expression
for(int i=0;i ^
Solution.java:18: error: ';' expected
inputList[i]=max
^
4 errors
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

3	0	3	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:01:21**
Average time taken between two compile attempts:	**00:00:27**
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

WriteX - Email Writing

62 / 100 | CEFR: **A2**

Question

Your name is Jessica Perez. You work at ValKount Inc., as a quality assurance engineer. The company development team requested a thorough test of their latest product, KwickSpek. After several rounds of testing, you observed 2 critical bugs:

1. Payment gateway errors
2. Error in capturing client details

Write an email to Nick Lewis (nick.lewis@valkount.com), the manager of the development team, clearly describing the bugs that you have identified. Also, ask the team to give the fix high priority, since the product launch is in 2 days.

Scores

Etiquette	83 / 100
Content	50 / 100
Grammar	54 / 100

Response

To: nick.lewis@valkount.com

Error Summary

● Email Etiquette 2

Subject: Critical bugs in KwickSpek

Hi nick,
During thorough testing , i dicovered two critical issues that requires immediate attenti
on. the first is a payment gatewat error where payment transactions are not being proc
essed correctly.The second is capturing client details resulting in incorrect details being
fetched from database.Both issues significantly afftect the experience of our clients.
As the product launch is just two days away, i strongly recommend resolving these bot
h issues as soon as possible to ensure a smooth delivery.
Regards
Jessica Perez,
Quality Assurance Engineer,ValKount.

●	Spelling	3
●	White Space	4
●	Style	0
●	Grammar	4
●	Typographical	2

Email Statistics

76	5	15	63	26
Total words	Total sentences	Average sentence length	Total unique words	Total stop words

Error Details

Email Etiquette

Hi nick,	Each word should start with a capitalized letter.
Regards Jessica Perez, Quality Assurance Engineer,ValKoun t.	Preferable to write each section of closing in new line.

Spelling

During thorough testing , i dicovered two critical issues tha t requires immed...	Possible spelling mistake found
...diate attention. the first is a payment gatewat error wher e payment transactions are no...	Possible spelling mistake found
...from database.Both issues significantly afftect the experi ence of our clients. As the p...	Possible spelling mistake found

White Space

During thorough testing , i dicovered two critical issues tha t re...	Put a space after the comma, but not before the comma
...tions are not being processed correctly.The second is cap turing client details resu...	Add a space between sentences
...ect details being fetched from database.Both issues signi ficantly afftect the experi...	Add a space between sentences

...ce of our clients. As the product launch is just two days a way, i strongly recomm...	Possible typo: you repeated a whitespace

Grammar

During thorough testing , i dicovered two critical issues tha t requires immediate attention.	Possible grammar error found. Consider replacing it with "required".
The second is capturing client details resulting in incorrect details being fetched from database.	Possible grammar error found. Consider inserting "the" over here.
As the product launch is just two days away, i strongly rec ommend resolving these both issues as soon as possible to ensure a smooth delivery.	Possible grammar error found. Consider removing "both" from here.
As the product launch is just two days away, i strongly rec ommend resolving these both issues as soon as possible to ensure a smooth delivery.	Possible grammar error found. Consider removing "a" from here.

Typographical

...sues that requires immediate attention. the first is a pay ment gatewat error where ...	This sentence does not start with an uppercase letter
The second is capturing client details resulting in incorrect details being fetched from database.	Possible typographical error found. Consider replacing it with "details,".

4 | Learning Resources

English Comprehension

- Learn about business e-mail etiquettes
- Learn about written english comprehension
- Learn about spoken english comprehension

Logical Ability

- Learn about how to solve problems by deriving complex rules
- Learn about syllogisms- the basics of deduction
- Practice examples of deductive reasoning

Quantitative Ability (Advanced)

- Learn about logarithms
- Learn about the concepts of probability and statistics
- Learn about permutations and combinations: from the shallows to the deep

WriteX - Email Writing

- How to improve email etiquette

Content

- Tips on writing a professional email

Grammar

- Email writing tips

Icon Index

Free Tutorial	Paid Tutorial	Youtube Video	Web Source
Wikipedia	Text Tutorial	Video Tutorial	Google Playstore