Rajeev Mukhiya

Test ID: 450014084000274 | 9472487029 | 6604548@rungta.org

Test Date: September 18, 2025

Logical Ability	53 /100
Data Science	67 /100
Computer Programming	79 /100
Computer Science (Level 3)	70 /100
Quantitative Ability (Advanced)	85 /100
English Comprehension	55 /100
WriteX - Essay Writing	61 /100
Automata	0 /100
Automata Fix	15 /100
Personality	✓ Completed

Logical Ability

53 / 100

Inductive Reasoning	50 / 100
Deductive Reasoning	53 / 100
Abductive Reasoning	56 / 100

Data Science

67 / 100

ML Algorithms and Implementation	100 / 100
Probability and Statistics	75 / 100
ML Experiments	0 / 100

Computer Programming

79 / 100

Basic Programming	Data Structures	OOP and Complexity Theory
79 / 100	**87** / 100	**72** / 100

Computer Science (Level 3)

70 / 100

C++	Formal Language and Automata Theory	Computer Networks
86 / 100	**83** / 100	**43** / 100

Quantitative Ability (Advanced)

85 / 100

Basic Mathematics	Advanced Mathematics	Applied Mathematics
83 / 100	**84** / 100	**89** / 100

English Comprehension

55 / 100 | CEFR: **B2**

Grammar	Vocabulary	Comprehension
55 / 100	**60** / 100	**51** / 100

WriteX - Essay Writing

61 / 100 | CEFR: **B1**

Content Score	Grammar Score
68 / 100	**44** / 100

Automata

0 / 100

Programming Ability	Programming Practices	Functional Correctness
0 / 100	**0** / 100	**0** / 100

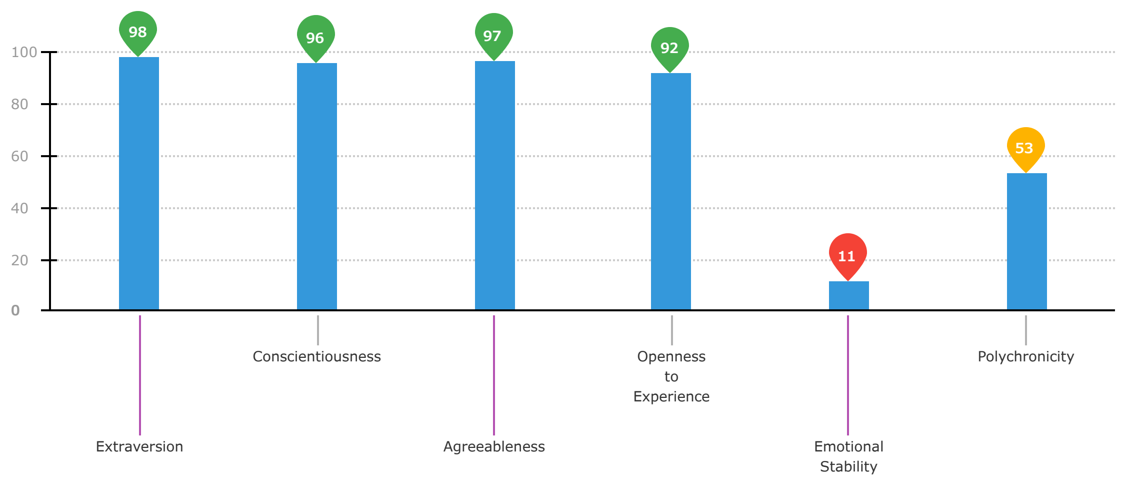
Automata Fix

15 / 100

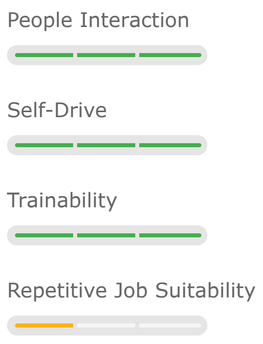
Logical Error	Syntactical Error	Code Reuse
25 / 100	**0** / 100	**0** / 100

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:

🟢 70 ≤ Score < 100

🟡 30 ≤ Score < 70

🔴 0 ≤ Score < 30

2 | Insights

English Comprehension

55 / 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

53 / 100

Inductive Reasoning

50 / 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

53 / 100

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

You are able to work out rules based on specific information and solve general work problems using these rules. This skill is required in data-driven research jobs where one needs to formulate new rules based on variable trends.

Abductive Reasoning

56 / 100

Quantitative Ability (Advanced)

85 / 100

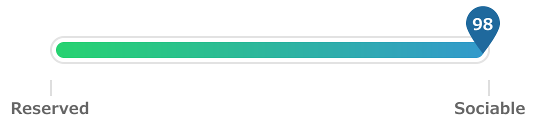
This test aims to measure your ability to solve problems on basic arithmetic operations, probability, permutations and combinations, and other advanced concepts.

You have excellent quantitative ability. You are able to apply mathematical concepts to real-life problems to analyze, solve and make predictions.

Personality

Competencies

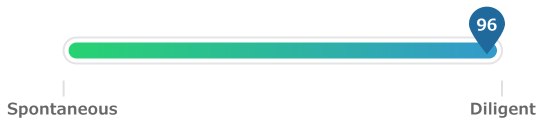
Extraversion



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 outgoing and seek out opportunities to meet new people.
- You tend to enjoy social gatherings and feels comfortable amongst strangers and friends equally.
- You display high energy levels and like to indulge in thrilling and exciting activities.
- You may tend to be assertive about your opinions and prefer action over contemplation.
- You take initiative and are more inclined to take charge than to wait for others to lead the way.
- Your personality is well suited for jobs demanding frequent interaction with people.

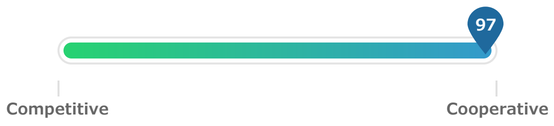
Conscientiousness



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

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

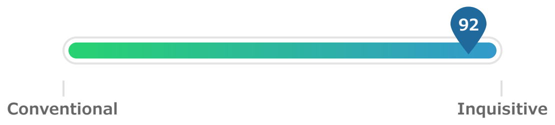
Agreeableness



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

- You are considerate and sensitive to the needs of others.
- You tend to put the needs of others ahead of your own.
- You are likely to trust others easily without doubting their intentions.
- You are compassionate and may be strongly affected by the plight of both friends and strangers.
- You are humble and modest and prefer not to talk about personal accomplishments.
- Your personality is more suitable for jobs demanding cooperation among employees.

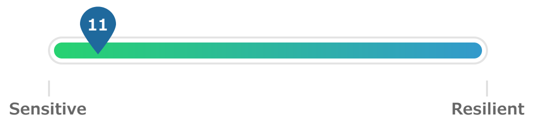
Openness to Experience



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

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

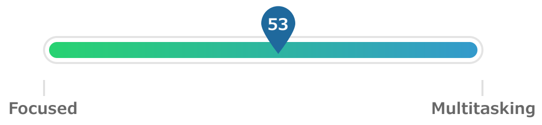
Emotional Stability



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

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

Polychronicity



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

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

Computer Science (Level 3)

● 70 / 100

This test aims to measures the knowledge and understanding on the concepts of C++, formal languages and automata theory and computer networks.

This test aims to measures the knowledge and understanding on the concepts of C++, formal languages and automata theory and computer networks.

- You have a sound knowledge of computer science and has a good understanding of the concepts, principles and theories.
- You are able to understand the practical applications of computer science and analyze real-world problems. You should focus on mastering niche topics and stay abreast with the latest developments.

Suggestion to Improve

\N

3 | Response

WriteX - Essay Writing

61 / 100 | CEFR: **B1**

Question

In your opinion, when is it justified for a company to fire its employees from the job? What are the considerations to be taken before making such a decision?
Support your response with reasons and suitable examples.

Scores

Content Score	Grammar Score
68 / 100	44 / 100

Response

justify reaspn to fire employees given below 1. Incompetence or poor performance ,es pacially after multiple warnings and opportunities to improve 2. Insubordination,such a s refusal to follow instructipn or companu rules. 3. If employee breaks important rules l ike being sishonest or stealing 4. If an employee shares private company information w ithout permission. 5 The compony should keep records of ehat the employee did wrong 6. The company should make sure the decision is fair and not vased on discrimination 7. Talkingto a human resources person or legan ecpert is good idea in difficult cases 8. If someonesis always late or does not come to work many times. 9. Comnsult eith hum an resources or legal experts for complex situation to minimize legal risk. 10. Review c ompany policies employment contracts, and comply witholocal lavor and anti-discrimin ation laws. 11. Make sure the reason for termination is not discriminatory or retaliator y; conduct investigations when needed(e,g,m in harasssnent cases.) 12. The employee get illegal money against compony 13. if an employee lies or cheats aout their work (fo r example write fake reports) 14, if an employee uses drugs or drinks alcohol while wor king which can be dangerous 15 if a personcauses safety problems at work, kike not fol lowing important safety 16. if an employee is not honest about their qualification or wo rk background.

Error Summary

Spelling	17
White Space	19
Style	0
Grammar	21
Typographical	2

Essay Statistics

218	20	11	159	65
Total words	Total sentences	Average sentence length	Total unique words	Total stop words

Error Details

Spelling

justify reaspn to fire employees given below ...	Possible spelling mistake found

justify reaspn to fire employees given below 1. Inco...	Possible spelling mistake found
... 1. Incompetence or poor performance ,espacially after multiple warnings and opportuniti...	Possible spelling mistake found
...uch as refusal to follow instructipn or companu rules. 3. If employee breaks important...	Possible spelling mistake found
...loyee breaks important rules like being sishonest or steal ing 4. If an employee shares ...	Possible spelling mistake found
... information without permission. 5 The compony should keep records of ehat the employe...	Possible spelling mistake found
.... 5 The company should keep records of ehat the employ ee did wrong 6. The company ...	Possible spelling mistake found
... make sure the decision is fair and not vased on discrimi nation 7. Talkingto a human...	Possible spelling mistake found
.... Talkingto a human resources person or legan ecpert is good idea in difficult cases ...	Possible spelling mistake found
...ingto a human resources person or legan ecpert is good i dea in difficult cases 8. If ...	Possible spelling mistake found
... is good idea in difficult cases 8. If someonesis always lat e or does not come to work ma...	Possible spelling mistake found
... does not come to work many times. 9. Comnsult eith hu man resources or legal experts ...	Possible spelling mistake found
...come to work many times. 9. Comnsult eith human reso urces or legal experts for co...	Possible spelling mistake found
...licies employment contracts, and comply witholocal lavor and anti-discrimination laws. 11...	Possible spelling mistake found
...oyment contracts, and comply witholocal lavor and anti-discrimination laws. 11. Make...	Possible spelling mistake found
...uct investigations when needed(e,g,m in harasssnent cas es.) 12. The employee get illegal ...	Possible spelling mistake found
...The employee get illegal money against compony 13. if a n employee lies or cheats aout...	Possible spelling mistake found
...pony 13. if an employee lies or cheats aout their work (f or example write fake rep...	Possible spelling mistake found

White Space

...sfy reaspn to fire employees given below 1. Incompetenc e or poor performance ,esp...	Possible typo: you repeated a whitespace
...ow 1. Incompetence or poor performance ,espacially afte r multiple warnings and o...	Put a space after the comma, but not before the comma
...le warnings and opportunities to improve 2. Insubordinat ion,such as refusal to fo...	Possible typo: you repeated a whitespace
...tunities to improve 2. Insubordination,such as refusal to follow instructipn or com...	Put a space after the comma

...t rules like being sishonest or stealing 4. If an employee shares private company...	Possible typo: you repeated a whitespace
...p records of ehat the employee did wrong 6. The compa ny should make sure the deci...	Possible typo: you repeated a whitespace
... is fair and not vased on discrimination 7. Talkingto a hu man resources person or...	Possible typo: you repeated a whitespace
...n ecpert is good idea in difficult cases 8. If someonesis al ways late or does not...	Possible typo: you repeated a whitespace
...lways late or does not come to work many times. 9. Com nsult eith human resource...	Possible typo: you repeated a whitespace
...t come to work many times. 9. Comnsult eith human res ources or legal experts fo...	Possible typo: you repeated a whitespace
...on is not discriminatory or retaliatory; conduct investigat ions when needed(e,g,m...	Possible typo: you repeated a whitespace
...y; conduct investigations when needed(e,g,m in harasss nent cases.) 12. The emplo...	Put a space after the comma
...in harasssnent cases.) 12. The employee get illegal mon ey against compony 13. i...	Possible typo: you repeated a whitespace
...loyee get illegal money against compony 13. if an emplo yee lies or cheats aout t...	Possible typo: you repeated a whitespace
...heats aout their work (for example write fake reports) 1 4, if an employee uses d...	Possible typo: you repeated a whitespace
...r work (for example write fake reports) 14, if an employe e uses drugs or drinks ...	Possible typo: you repeated a whitespace
...hol while working which can be dangerous 15 if a person causes safety problems at...	Possible typo: you repeated a whitespace
...while working which can be dangerous 15 if a personcau ses safety problems at wor...	Possible typo: you repeated a whitespace
...ork, kike not following important safety 16. if an employ ee is not honest about t...	Possible typo: you repeated a whitespace

Grammar

justisfy reaspn to fire employees given below 1.	Possible grammar error found. Consider replacing it with "for".
justisfy reaspn to fire employees given below 1.	Possible grammar error found. Consider inserting "is" over here.
Incompetence or poor performance ,espacially after multipl e warnings and opportunities to improve 2.	Possible grammar error found. Consider replacing it with "performance,".
Incompetence or poor performance ,espacially after multipl e warnings and opportunities to improve 2.	Possible grammar error found. Consider replacing it with "especially".
If employee breaks important rules like being sishonest or stealing 4.	Possible grammar error found. Consider inserting "an" over here.

The company should make sure the decision is fair and not vased on discrimination 7.	Possible grammar error found. Consider replacing it with "discrimination.".
Talkingto a human resources person or legan ecpert is goo d idea in difficult cases 8.	Possible grammar error found. Consider inserting "a" over here.
Talkingto a human resources person or legan ecpert is goo d idea in difficult cases 8.	Possible grammar error found. Consider replacing it with "cases.".
If someoneis always late or does not come to work many times.	Possible grammar error found. Consider inserting "is" over here.
Comnsult eith human resources or legal experts for comple x situation to minimize legal risk.	Possible grammar error found. Consider replacing it with "situations".
Review company policies employment contracts, and compl y witholocal lavor and anti-discrimination laws.	Possible grammar error found. Consider replacing it with "policies,".
The employee get illegal money against compony 13.	Possible grammar error found. Consider replacing it with "got".
The employee get illegal money against compony 13.	Possible grammar error found. Consider replacing it with "from".
The employee get illegal money against compony 13.	Possible grammar error found. Consider inserting "the" over here.
The employee get illegal money against compony 13.	Possible grammar error found. Consider replacing it with "company.".
if an employee lies or cheats aout their work (for example write fake reports) 14, if an employee uses drugs or drinks alcohol while working which can be dangerous 15 if a perso ncauses safety problems at work, kike not following import ant safety 16.	Possible grammar error found. Consider replacing it with "work,".
if an employee lies or cheats aout their work (for example write fake reports) 14, if an employee uses drugs or drinks alcohol while working which can be dangerous 15 if a perso ncauses safety problems at work, kike not following import ant safety 16.	Possible grammar error found. Consider replacing it with "writing".
if an employee lies or cheats aout their work (for example write fake reports) 14, if an employee uses drugs or drinks alcohol while working which can be dangerous 15 if a perso ncauses safety problems at work, kike not following import ant safety 16.	Possible grammar error found. Consider replacing it with "reports,".
if an employee lies or cheats aout their work (for example write fake reports) 14, if an employee uses drugs or drinks alcohol while working which can be dangerous 15 if a perso ncauses safety problems at work, kike not following import ant safety 16.	Possible grammar error found. Consider replacing it with "an".
if an employee lies or cheats aout their work (for example write fake reports) 14, if an employee uses drugs or drinks alcohol while working which can be dangerous 15 if a perso ncauses safety problems at work, kike not following import ant safety 16.	Possible grammar error found. Consider replacing it with "safety.".

if an employee is not honest about their qualification or wo rk background.	Possible grammar error found. Consider replacing it with "qualifications".

Typographical

... get illegal money against compony 13. if an employee li es or cheats aout their w...	This sentence does not start with an uppercase letter
...ike not following important safety 16. if an employee is n ot honest about their q...	This sentence does not start with an uppercase letter

Automata

0 / 100 Code Replay

Question 1 (Language: C)

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

0 / 100

Programming ability score cannot be generated. This is because source code has syntax/ runtime errors and is unparseable.

Programming Practices

0 / 100

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

Functional Correctness

0 / 100

Syntactically incorrect code. The source code has syntax errors in it.

Final Code Submitted **Compilation Status: Fail**

```
//Header Files
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
```

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.

```
 6
 7 typedef struct{
 8  char *data;
 9  int size;
10 } array_single_char;
11
12 int calculateDesktopProductIDs(array_single_char productIDs){
13     char desktopIDs[]= "CDEGHIJKLNMO";
14   int count =0;
15   for(i=0; i<productIDs.size; i++){
16       char ch=toupper(productIDs.data[i]);
17       for( int j=0; desktopIDs[j] != '\0'; j++){
18           if (ch ==desktopIDs[j]){
19               count++;
20               break;
21           }
22       }
23   }
24   return count
25 }
26
27     int man(){
28         array_single_char productIDs;
29         scanf("%d". &productIDs.size);
30         productIDs.data =(char *)malloc(sizeop(char) * productIDs.size);
31
32         fpr (int idx =0; idx<productIDs.size; idx++){
33             scanf("%c", &productIDs.data[idx])
34         }
35
36         int resuult = calculateDesktopProductIDs(productIDs);
37         printf("%d\n",result);
38         free(productID.data);
39         return 0;
40     }
41
42
```

Best case code: O(N)

*N represents number of products.

Errors/Warnings

```
Compiling failed with exitcode 1, compiler output:
source_5811.c: In function
'calculateDesktopProductIDs':
source_5811.c:15:7: error: 'i' undeclared (first use in
this function)
for(i=0; i ^
source_5811.c:15:7: note: each undeclared identifier
is reported only once for each function it appears in
source_5811.c:16:15: warning: implicit declaration of
function 'toupper' [-Wimplicit-function-declaration]
char ch=toupper(productIDs.data[i]);
^
source_5811.c:25:1: error: expected ';' before '}'
token
}
^
source_5811.c: In function 'man':
source_5811.c:29:21: error: expected identifier before
'&' token
scanf("%d". &productIDs.size);
^
source_5811.c:30:41: warning: implicit declaration of
function 'sizeop' [-Wimplicit-function-declaration]
productIDs.data =(char *)malloc(sizeop(char) *
productIDs.size);
^
source_5811.c:30:48: error: expected expression
before 'char'
productIDs.data =(char *)malloc(sizeop(char) *
productIDs.size);
^
source_5811.c:32:10: warning: implicit declaration of
function 'fpr' [-Wimplicit-function-declaration]
fpr (int idx =0; idx ^
source_5811.c:32:15: error: expected expression
before 'int'
fpr (int idx =0; idx ^
source_5811.c:32:54: error: expected ';' before '{'
token
fpr (int idx =0; idx ^
source_5811.c:37:24: error: 'result' undeclared (first
use in this function)
printf("%d\n",result);
^
source_5811.c:38:15: error: 'productID' undeclared
(first use in this function)
free(productID.data);
^
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

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

Response time:	**00:31:01**
Average time taken between two compile attempts:	**00:10:20**
Average test case pass percentage per compile:	**8.89%**

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: C)

In an online word recognition game for kids, the user needs to find the number of times the given word occurs in the sentence. Both the given word and the sentence displayed on the user interface consist of letters from the English alphabet only and are case insensitive (i.e., ‘toddler’ is same as ‘Toddler’). Neither the word nor the sentence contain any white-spaces or special symbols.

Write an algorithm to print the number of times the given word appears in the sentence.

Scores

Programming Ability

0 / 100

NA

Programming Practices

0 / 100

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

Functional Correctness

0 / 100

NA

Final Code Submitted

Compilation Status: Pass

```
//Header Files
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>

/* only used in string related operations */
typedef struct String string;
struct String
{
    char *str;
};

char *input(FILE *fp, int size, int has_space)
{
    int actual_size = 0;
    char *str = (char *)malloc(sizeof(char)*(size+actual_size));
    char ch;
    if(has_space == 1)
    {
        while(EOF != (ch=fgetc(fp)) && ch != '\n')
        {
            str[actual_size] = ch;
            actual_size++;
            if(actual_size >= size)
            {
                str = realloc(str,sizeof(char)*actual_size);
            }
        }
    }
    else
```

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 length of the string

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

```
    {
        while(EOF != (ch=fgetc(fp)) && ch != '\n' && ch != ' ')
        {
            str[actual_size] = ch;
            actual_size++;
            if(actual_size >= size)
            {
                str = realloc(str,sizeof(char)*actual_size);
            }
        }
    }
    actual_size++;
    str = realloc(str,sizeof(char)*actual_size);
    str[actual_size-1] = '\0';
    return str;
}
/* only used in string related operations */


/*
 * The first line of the input consists of a string- sentence, representing the sentence on the user interface.
The second line consists of a string- word, representing the given word.
 */
int  countWords(string sentence, string word)
{
    int  answer;
    // Write your code here


    return answer;
}

int main()
{
    string sentence;
    string word;


    //input for sentence
    sentence.str = input(stdin, 100, 1);


    //input for word
    word.str = input(stdin, 100, 1);


    int result = countWords(sentence, word);
    printf("%d", result);
    return 0;
}
```

Test Case Execution

Passed TC: **0%**

Total score 0/16

0% Basic(**0**/10)

0% Advance(**0**/5)

0% Edge(**0**/1)

Compilation Statistics

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

Response time:	**00:00:15**
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

Automata Fix

15 / 100 Code Replay

Question 1 (Language: C)

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

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

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

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

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
void drawPrintPattern(int num)
{
  int i,j,print = 1;
  for(i=1;i<=num;i++)
  {
    // for(j=1;j<=2*i;j++);
    {
      printf("%d ",print);
    }
    // printf("\n");
  }
}

```

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: **12.5%**

Total score 1/8

0%	0%	100%
Basic(**0**/7)	Advance(**0**/0)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:04:40**
Average time taken between two compile attempts:	**00:00:47**
Average test case pass percentage per compile:	**4.2%**

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: C)

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

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

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

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

and hence should not be counted.

Scores

Final Code Submitted

Compilation Status: Pass

```
// You can print the values to stdout for debugging
int sameElementCount(int size, int *inputList)
{
   int i,count =0;
   for(i=0;i<size-1;i++)
   {
     if((inputList[i]%2==0)&&(inputList[i]==inputList[i+1]))
        count++;
   }
   return count;
}


```

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(**7**/7)	Advance(**0**/0)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:02:07**
Average time taken between two compile attempts:	**00:02:07**
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 3 (Language: C)

The function/method ***manchester*** accepts two arguments - *len* and *arr,* an integer representing the length of the list (*len* ≥ 0) and a list of integers, respectively*.* Each element of *arr* represents a bit - 0 or 1. The output is a list with the following property: for each element in the input list *arr*, if the bit arr[i] is the same as *arr*[i-1], then the element of the output list is 0. If they are different, then its 1. For the first bit in the input list, assume its previous bit to be 0. This encoding is stored and returned in a new list.

For e.g. if *arr* is {0,1,0,0,1,1,1,0}, the function/method should return an list {0,1,1,0,1,0,0,1}.

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

Scores

Final Code Submitted **Compilation Status: Pass**

```c
// You can print the values to stdout for debugging
#include<stdio.h>
#include <stdlib.h>
void manchester(int len, int* arr)
{
   int* res = (int*)malloc(sizeof(int)*len);

   if (res==NULL){
     return;
   }
   res[0] = arr[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

```
    for(int i = 1; i < len; i++){
      res[i] = (arr[i]==arr[i-1]);
    }
    for(int i =0; i<len; i++)
        printf("%d ",res[i]);
}
```

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: **20%**

Total score 1/5

50%	**0%**	**0%**
Basic(**1**/2)	Advance(**0**/3)	Edge(**0**/0)

Compilation Statistics

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

Response time:	**00:03:10**
Average time taken between two compile attempts:	**00:01:35**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 4 (Language: C)

The function/method ***multiplyNumber*** returns the multiplicative product of the maximum two of three input numbers.

The function/method accepts three integers *numA*, *numB* and *numC*, representing the input numbers.

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

Scores

Final Code Submitted **Compilation Status: Fail**

```
 1 // You can print the values to stdout for debugging
 2 int multiplyNumber(int numA, int numB, int numC)
 3 {
 4   int result,min,max,mid;
 5   max=(numA>numB)?numA>numC)?numA:numC):(numB>numC)?
   numB:numC);
 6   min=(numA<numB)?((numA<numC)?numA:numC):((numB<num
   C)?numB:numC);
 7   mid=(numA+numB+numC)-(min+max);
 8   result=(max*int mid);
 9   return result;
10 }
```

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

```
In file included from main_31.c:3:
source_31.c: In function 'multiplyNumber':
source_31.c:5:29: error: expected ':' before ')' token
max=(numA>numB)?numA>numC)?numA:numC):
(numB>numC)?numB:numC);
^
:
source_31.c:5:29: error: expected statement before
')' token
source_31.c:5:30: error: expected expression before
'?' token
max=(numA>numB)?numA>numC)?numA:numC):
(numB>numC)?numB:numC);
^
source_31.c:5:40: error: expected statement before
')' token
max=(numA>numB)?numA>numC)?numA:numC):
(numB>numC)?numB:numC);
^
source_31.c:5:41: error: expected expression before
':' token
max=(numA>numB)?numA>numC)?numA:numC):
(numB>numC)?numB:numC);
^
source_31.c:5:63: error: expected statement before
')' token
max=(numA>numB)?numA>numC)?numA:numC):
(numB>numC)?numB:numC);
^
source_31.c:8:16: error: expected expression before
```

```
'int'
result=(max*int mid);
^~~
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

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

Response time:	**00:00:07**
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 5 (Language: C)

The function/method ***median*** accepts two arguments - *size* and *inputList*, an integer representing the length of a list and a list of integers, respectively. It is supposed to calculate and return the median of elements in the input list.

However, the function/method **median** works only for odd-length lists because of incomplete code.

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

(Please refer to the Helper Code tab for details regarding these functions/methods.)

.

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
float median(int size, int * inputList)
{
  int start_index = 0;
  int end_index = size-1;
  float res = -1;
  if(size%2!=0) // odd size inputList
  {
    int median_order = ((size+1)/2);
    res = (float)quick_select(inputList, start_index, end_index, median_order);
  }
  else // even size inputList
  {
     // Write code here
  }
  return res;
}


```

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: **42.86%**

Total score 3/7

50%	0%	100%
Basic(**2**/4)	Advance(**0**/2)	Edge(**1**/1)

Compilation Statistics

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

Response time:	00:00:00
Average time taken between two compile attempts:	00:00:00
Average test case pass percentage per compile:	42.9%

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: C)

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

You must implement the function/method *isRightTriangle* accepts three points - *P1, P2, P3* as input and checks whether the given three points can make a right angle triangle.

If they make a right angle triangle, the function/method returns 1. Otherwise, it returns 0.
(Please refer to the Helper Code tab for details regarding the structure of Point and the predefined functions/methods around it).

Scores

Final Code Submitted **Compilation Status: Fail**

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
   // write your code here
}


```

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

```
In file included from main_23.c:5:
source_23.c: In function 'isRightTriangle':
source_23.c:5:1: error: control reaches end of non-void function [-Werror=return-type]
}
^
cc1: some warnings being treated as errors
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

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

Response time:	**00:00:01**
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: C)

The function/method **countOccurrence** accepts three arguments - *arr*, an integer list*, len*, an integer representing the size of the list, and *value,* an integer. It is supposed to return the count of occurrences of *value* in the input list *arr* of size *len* (0 ≤ *len*).

The function/method 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**

```c
// You can print the values to stdout for debugging
int countOccurrence( int len, int value, int *arr)
{
  int i=0, count = 0;
  while(i<len){
    if(arr[i]==value)
      count += 1;
  }
  return count;
}

```

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: **11.11%**

Total score 1/9	**0%** Basic(**0**/3)	**0%** Advance(**0**/5)	**100%** Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:00:06**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**11.1%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

4 | Learning Resources

English Comprehension

Learn about business e-mail etiquettes	🏷	🌐	📖
Learn about written english comprehension	🏷	🌐	🎬
Learn about spoken english comprehension	🏷	🌐	🎬

Logical Ability

Learn about the art of deduction	$	🌐	📖
Learn about the fallacies in deductive reasoning	🏷	🌐	📖
Learn about validity of arguments	🏷	▶	🎬

Quantitative Ability (Advanced)

Learn about Fermet's last theorem	🏷	W	📖
Learn about the real life applications of probability	🏷	🌐	📖
Learn about the application of Bayes' Theorem in varied fields	🏷	W	📖

Icon Index

🏷	Free Tutorial	$	Paid Tutorial	▶	Youtube Video	🌐	Web Source
▶	Wikipedia	📖	Text Tutorial	🎬	Video Tutorial	▷	Google Playstore