Sumehar Singh Grewal

Test ID: 450015228581986 | 8360645330 | sumeharsinghgrewal@gmail.com

Test Date: November 29, 2025

Computer Science	Logical Ability	Computer Programming	Quantitative Ability (Advanced)
77 /100	56 /100	59 /100	48 /100

English Comprehension	Automata Fix	Automata Pro	Personality
64 /100	58 /100	51 /100	Completed

Computer Science — 77 / 100

OS and Computer Architecture	DBMS	Computer Networks
65 / 100	95 / 100	89 / 100

Logical Ability — 56 / 100

Inductive Reasoning	Deductive Reasoning	Abductive Reasoning
51 / 100	61 / 100	55 / 100

Computer Programming — 59 / 100

Basic Programming	Data Structures	OOP and Complexity Theory
57 / 100	63 / 100	58 / 100

Quantitative Ability (Advanced)

48 / 100

Basic Mathematics	Advanced Mathematics	Applied Mathematics
44 / 100	49 / 100	50 / 100

English Comprehension

64 / 100 | CEFR: **B2**

Grammar	Vocabulary	Comprehension
65 / 100	61 / 100	66 / 100

Automata Fix

58 / 100

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

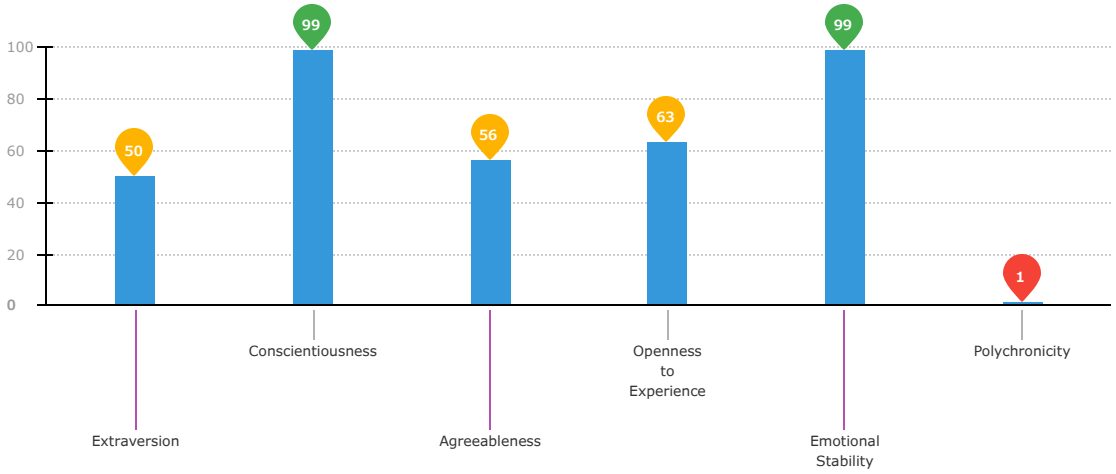
Automata Pro

51 / 100

Programming Practices	Functional Correctness
100 / 100	29 / 100

Personality

Completed



Competencies

People Interaction

Self-Drive

Trainability

Repetitive Job Suitability

Work attributes

1 | Introduction

About the Report

This report provides a detailed analysis of the candidate's performance on different assessments. The tests for this job role were decided based on job analysis, O*Net taxonomy mapping and/or criterion validity studies. The candidate’s responses to these tests help construct a profile that reflects her/his likely performance level and achievement potential in the job role

This report has the following sections:

The **Summary** section provides an overall snapshot of the candidate’s performance. It includes a graphical representation of the test scores and the subsection scores.

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

The **Response** section captures the response provided by the candidate. This section includes only those tests that require a subjective input from the candidate and are scored based on artificial intelligence and machine learning.

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

Score Interpretation

All the test scores are on a scale of 0-100. All the tests except personality and behavioural evaluation provide absolute scores. The personality and behavioural tests provide a norm-referenced score and hence, are percentile scores. Throughout the report, the colour codes used are as follows:

- 70 ≤ Score < 100
- 30 ≤ Score < 70
- 0 ≤ Score < 30

2 | Insights

English Comprehension

64 / 100 | CEFR: **B2**

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

You have a good understanding of commonly used grammatical constructs. You are able to read and understand articles, reports and letters/mails related to your day-to-day work. The ability to read, understand and interpret business-related documents is essential in most jobs, especially the ones that involve research, technical reading and content writing.

Logical Ability

56 / 100

Inductive Reasoning

51 / 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.

Deductive Reasoning

61 / 100

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

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

Abductive Reasoning

55 / 100

Quantitative Ability (Advanced)

48 / 100

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

You are good at basic arithmetic. You are able to solve real-world problems that involve simple addition, subtraction, multiplication and division.

Personality

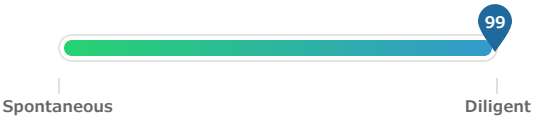
Competencies

Extraversion

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

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

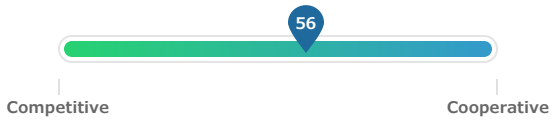
Conscientiousness



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

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

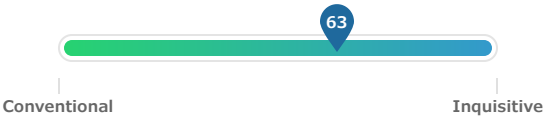
Agreeableness



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

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

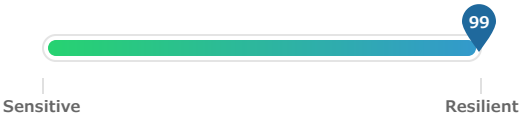
Openness to Experience



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

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

Emotional Stability



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

- You are calm and composed in nature.
- You tend to maintain composure during high pressure situations.
- You are very confident and comfortable being yourself.
- You find it easy to resist temptations and practice moderation.
- You are likely to remain emotionally stable in jobs with high stress levels.

Polychronicity

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

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

3 | Response

Automata Pro

51 / 100 | Code Replay

Question 1 (Language: C++20)

A child's parent goes for a jog every morning. The child follows the parent several minutes later. The parent starts at a position that is X_1meters away from their home and runs in a straight line at a constant speed of V_1meters per step for N steps.

The child is standing X_2metres away from their home. They wonder how fast they must run at a constant speed of V_2metres per step to achieve a maximum F, where F equals the number of their parent's footsteps that the child will land on during their run. The first step that the child will land on from their starting position will have been landed on by their parent.

Note that, if more than one prospective speed results in the same number of maximum common steps, output the highest prospective speed as V_2.

Write an algorithm to calculate F and V_2.

Scores

Programming Practices

100 / 100

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

Functional Correctness

36 / 100

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

Final Code Submitted — **Compilation Status: Pass**

```
1  // Sample code to read input and write output:
2
3  /*
4  #include <iostream>
5
6  using namespace std;
7
8  int main()
9  {
10   char name[20];
11   cin >> name;                // Read input from STDIN
12   cout << "Hello " << name;       // Write output to STDOUT
13   return 0;
14 }
15 */
```

Code Analysis

Average-case Time Complexity

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

Best case code: O(N logN)

*N represents number of steps for which father runs.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

```
16
17 // Warning: Printing unwanted or ill-formatted data to output will c
   ause the test cases to fail
18
19 #include <iostream>
20
21 using namespace std;
22
23 int main()
24 {
25    // Write your code here
26    int parentPos,childPos,velParent,steps;
27    cin>>parentPos;
28    cin>>childPos;
29    cin>>velParent;
30    cin>>steps;
31    int f,velChild;
32    if((parentPos-childPos)==velParent)
33    {
34       f = ((steps*velParent)+(velParent))/velParent;
35       velChild = velParent;
36       cout<<f<<" "<<velChild;
37    }
38    else if(parentPos-childPos==1){
39       f = steps+1;
40       velChild = 1;
41       cout<<f<<" "<<velChild;
42    }
43    else{
44       f = 1;
45       velChild = (velParent*steps+parentPos)-childPos;
46       cout<<f<<" "<<velChild;
47    }
48    return 0;
49 }
```

Readability & Language Best Practices

Line 31: Variables are given very short name.

Test Case Execution Passed TC: **50%**

Total score 5/10

83%	0%	0%
Basic(**5**/6)	Advance(**0**/3)	Edge(**0**/1)

Compilation Statistics

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

Response time:	**00:29:21**
Average time taken between two compile attempts:	**00:09:47**
Average test case pass percentage per compile:	**36.67%**

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++20)

A square matrix A[1..n][1..n] is called palindromic if A[i][j] = A[n + 1 - i][n + 1 - j] for all 1 ≤ i, j ≤ n.

Given a matrix *inputMat*[1..N][1..M], find the number of elements in its largest palindromic square sub-matrix.

Scores

Programming Practices

100 / 100

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

Functional Correctness

22 / 100

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

Final Code Submitted **Compilation Status: Pass**

```
1  // Sample code to read input and write output:
2
3  /*
4  #include <iostream>
5
6  using namespace std;
7
8  int main()
9  {
10   char name[20];
11   cin >> name;                  // Read input from STDIN
12   cout << "Hello " << name;      // Write output to STDOUT
13   return 0;
14 }
15 */
16
17 // Warning: Printing unwanted or ill-formatted data to output will c
   ause the test cases to fail
18
19 #include <iostream>
20
21 using namespace std;
22
23 int main()
24 {
25   // Write your code here
26   int n,m;
27   cin>>n>>m;
28   int matrix[n][m];
29   for(int i=0;i<n;i++)
30   {
31     for(int j=0;j<m;j++)
32     {
33       cin>>matrix[i][j];
34     }
35   }
36   int count=0;
37   int maxi=0;
38   if(n<m){
39     maxi = m;
40   }
41   else{
42     maxi =n;
43   }
44   for(int i=0;i<n;i++)
45   {
46     for(int j=0;j<m;j++)
47     {
48       if(matrix[i][j]==matrix[maxi-3-i][maxi-3-j])
```

Code Analysis

Average-case Time Complexity

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

Best case code: $O(N^2)$

*N represents number of rows or number of columns

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

Readability & Language Best Practices

Line 26: Variables are given very short name.

```
49          {
50              count++;
51          }
52      }
53  }
54  cout<<count;
55  return 0;
56 }
```

Test Case Execution

Passed TC: **15.38%**

Total score

2/13

14%	20%	0%
Basic(**1**/7)	Advance(**1**/5)	Edge(**0**/1)

Compilation Statistics

10	9	1	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:29:50**
Average time taken between two compile attempts:	**00:02:59**
Average test case pass percentage per compile:	**11.54%**

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: C++)

The function/method ***manchester*** print space-separated integers with the following property: for each element in the input array *arr*, a counter is incremented if the bit *arr*[i] is the same as *arr*[i-1]. Then the increment counter value is added to the output array to store the result.
If the bit *arr*[i] and *arr*[i-1] are different, then 0 is added to the output array. For the first bit in the input array, assume its previous bit to be 0. For example, if *arr* is {0,1,0,0,1,1,1,0}, the function/method should print 1 0 0 2 0 3 4 0.

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

The function/method ***manchester*** 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**

```
1  #include<iostream>
2  using namespace std;
3  void manchester(int size, int* arr)
4  {
5     bool result;
6     int count  =0;
7     int* res = new int[size];
8     for(int i = 0; i < size; i++)
9     {
10       if(i==0)
11          result= (arr[i]==1);
12       else
13          result = (arr[i]==arr[i-1]);
14       res[i] = (result)?(0):(++count);
15    }
16    for(int i=0; i<size; i++)
17    {
18       cout<<res[i]<<" ";
19    }
20 }
```

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: **14.29%**

Total score

1/7

0%	0%	100%
Basic(**0**/5)	Advance(**0**/1)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:05:52**
Average time taken between two compile attempts:	**00:01:57**
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 2 (Language: C++)

The function/method ***printFibonacci*** accepts an integer *num,* representing a number.
The function/method ***printFibonacci*** prints first *num* numbers of the Fibonacci series.
For example, given input 5, the function should print the string “0 1 1 2 3” (without quotes).

The function/method compiles successfully but fails to give 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

```
1  #include<iostream>
2  using namespace std;
3  void printFibonacci(int num)
4  {
5     long num1 = 0;
6     long num2 = 1;
7     for (int i = 1; i <= num; ++i)
8     {
9        cout<<num1<<" ";
10       long sum = num1 + num2;
11       num1 = num2;
12       num2 = sum;
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

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%	100%	100%
Basic(**5**/5)	Advance(**2**/2)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:01:48**
Average time taken between two compile attempts:	**00:00:36**
Average test case pass percentage per compile:	**33.3%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 3 (Language: C++)

The function/method ***reverseHalfArray*** modify the input list by reversing the input list from the second half.
For example, if the *inputList* is [20, 30, 10, 40, 50], the function/method is expected to modify the *inputList* like [20, 30, 50, 40, 10].

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

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

```
1  void reverseHalfArray(int size, int *inputList)
2  {
3     int i, temp;
4     for(i=size/2; i< size ; i++)
5     {
6        temp = inputList[size-1];
7        inputList[size-1] = inputList[i];
8        inputList[i] = temp;
9        size -= 1;
10    }
11 }
```

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%	100%	100%
Basic(**4**/4)	Advance(**2**/2)	Edge(**2**/2)

Compilation Statistics

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

Response time:	**00:00:59**
Average time taken between two compile attempts:	**00:00:30**
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: C++)

The function/method ***printPattern*** accepts an argument *num,* an integer.

The function/method ***printPattern*** print the first *num* lines of the pattern as shown below.
For example, if *num* = 3, the pattern should be:
1 1
2 2 2 2
3 3 3 3 3 3

The function/method ***printPattern*** compiles successfully but fails to print 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**

```
1  #include<iostream>
2  using namespace std;
3  void printPattern(int num)
4  {
5     int i,j;
6     for(i=1;i<=num;i++)
7     {
8        for(j=1;j<=2*i;j++)
9        {
10          cout<<i<<" ";
11       }
12       cout<<"\n";
13    }
14 }
15
```

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

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

Response time:	**00:01:43**
Average time taken between two compile attempts:	**00:00:34**
Average test case pass percentage per compile:	**33.3%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 5 (Language: C++)

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 the time and is already implemented in the default code (Do not write this definition again in your code):

```
class Time

{

   int hour;

   int minute;

   int second;

   int Time :: Time_compareTo( Time* time2)

   {

      /*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) */

   }

   void Time :: Time_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: Fail**

```
1 // You can print the values to stdout for debugging
2 #include<iostream>
3 using namespace std;
4 int difference_in_times(Time *time1, Time *time2)
5 {
6
```

Code Analysis

Average-case Time Complexity

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

```
7    // write your code here
8 }
9
```

Best case code:

*N represents

Errors/Warnings

```
In file included from main_24.cpp:8:
source_24.cpp: In function 'int
difference_in_times(Time*, Time*)':
source_24.cpp:8:1: error: no return statement in
function returning non-void [-Werror=return-type]
}
^
cc1plus: some warnings being treated as errors
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

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

Response time:	**00:01:26**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 6 (Language: C++)

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

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

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

Helper Description
The following class is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
class Point

{

   private:

   int X;

   int Y;

   double Point_calculateDistance(Point *point1, Point *point2)

   {

      /*Return the euclidean distance between two input points.
```

```
    This can be called as -

    *  If P1 and P2 are two points then -

    *  P1->Point_calculateDistance(P2);*/

  }

}
```

Scores

Final Code Submitted — Compilation Status: Fail

```
1  // You can print the values to stdout for debugging
2  #include<iostream>
3  using namespace std;
4
5  int isRightTriangle(Point *P1, Point *P2, Point *P3)
6  {
7     // write your code here
8  double side1 = P1->Point_calculateDistance(P2);
9  double side2 = P2->Point_calculateDistance(P3);
10 double side3 = P3->Point_calculateDistance(P1);
11 if(side1+side2<side3||side1+side3<side2||side3+side2<side1)
12 {
13
14 }
15
16 }
```

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.cpp:8:
source_23.cpp: In function 'int isRightTriangle(Point*, Point*, Point*)':
source_23.cpp:16:1: error: no return statement in function returning non-void [-Werror=return-type]
}
^
cc1plus: some warnings being treated as errors
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

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

Response time:	**00:06:59**
Average time taken between two compile attempts:	**00:06:59**
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 ***replaceMinMax*** is supposed to replace all the even elements of the input list with the maximum element of the list, also replace all the odd elements of arr with the minimum element of the list.

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

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

Scores

Final Code Submitted | **Compilation Status: Pass**

Code Analysis

```
// You can print the values to stdout for debugging
#include<iostream>
void replaceMinMax(int size, int* arr)
{
   int i=0;
   if(size>0)
   {
      int max = arr[0];
      int min = arr[0];
      for(i=0;i<size;++i)
      {
         if(max<arr[i])
         {
            max = arr[i];
         }
         else if(min > arr[i])
         {
            min = arr[i];
         }
      }
      for(i=0;i<size;++i)
      {
         if(arr[i] % 2 == 0)
            arr[i]=max;
         else
            arr[i]=min;
      }
   }
}
```

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

10/10

100%	100%	100%
Basic(**3**/3)	Advance(**5**/5)	Edge(**2**/2)

Compilation Statistics

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

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

4 | Learning Resources

English Comprehension

- Improve your hold on the language by reading Shakespearan plays
- Learn about how to get better at reading
- Read opinions to improve your comprehension

Logical Ability

- Learn about validity of arguments
- Practice Sherlock Holmes' puzzles and develop your deductive logic
- Practice your Inductive Reasoning Skills!

Quantitative Ability (Advanced)

- Learn about percentages
- Learn about simple and compount interests
- Watch a video on time, speed and distance

Icon Index

Free Tutorial | Paid Tutorial | Youtube Video | Web Source

Wikipedia | Text Tutorial | Video Tutorial | Google Playstore