

# Radhika Bhati

**Test ID:** 432008073000236 | 9871043466 | radhika.bhati.22cse@bmu.edu.in

**Test Date:** November 23, 2024

<b>Computer Science</b>  <b>78</b> /100	<b>Logical Ability</b>  <b>63</b> /100	<b>Computer Programming</b>  <b>61</b> /100	<b>Quantitative Ability (Advanced)</b>  <b>56</b> /100
<b>English Comprehension</b>  <b>71</b> /100	<b>WriteX - Essay Writing</b>  <b>83</b> /100	<b>Automata Fix</b>  <b>43</b> /100	<b>Automata Pro</b>  <b>96</b> /100
<b>Personality</b>  <b>Completed</b>			

<b>Computer Science</b>			<b>78</b> / 100
OS and Computer Architecture	DBMS	Computer Networks	
<b>91</b> / 100	<b>67</b> / 100	<b>56</b> / 100	

<b>Logical Ability</b>			<b>63</b> / 100
Inductive Reasoning	Deductive Reasoning	Abductive Reasoning	
<b>64</b> / 100	<b>64</b> / 100	<b>62</b> / 100	

## Computer Programming

61 / 100

Basic Programming

62 / 100

Data Structures

61 / 100

OOP and Complexity Theory

61 / 100

## Quantitative Ability (Advanced)

56 / 100

Basic Mathematics

49 / 100

Advanced Mathematics

59 / 100

Applied Mathematics

60 / 100

## English Comprehension

71 / 100

CEFR: **C1**

Grammar

73 / 100

Vocabulary

74 / 100

Comprehension

67 / 100

## WriteX - Essay Writing

83 / 100

CEFR: **C1**

Content Score

86 / 100

Grammar Score

78 / 100

## Automata Fix

43 / 100

Logical Error

50 / 100

Code Reuse

0 / 100

Syntactical Error

100 / 100

## Automata Pro

96 / 100

Programming Ability

90 / 100

Programming Practices

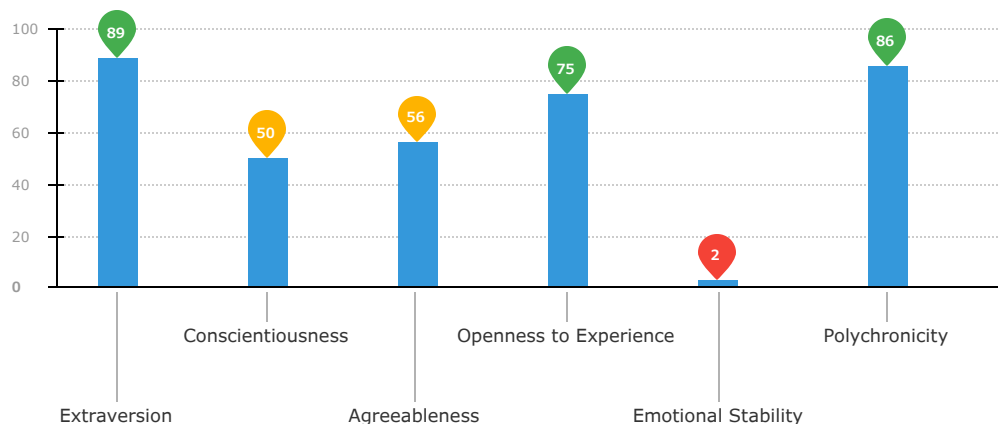
100 / 100

Functional Correctness

68 / 100

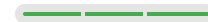
# Personality

Completed

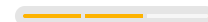


## Competencies

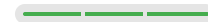
People Interaction



Self-Drive



Trainability



Repetitive Job Suitability



## Work attributes

## 1 | Introduction

### About the Report

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

This report has the following sections:

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

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

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

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

### Score Interpretation

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

- Scores between 67 and 100
- Scores between 33 and 67
- Scores between 0 and 33

## 2 | Insights

### English Comprehension

71 / 100

CEFR: **C1**

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

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

### Logical Ability

63 / 100



#### Inductive Reasoning

64 / 100

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

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



#### Deductive Reasoning

64 / 100

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

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



#### Abductive Reasoning

62 / 100

### Quantitative Ability (Advanced)

56 / 100

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

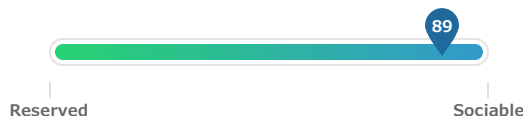
You are able to solve word problems on basic concepts of percentages, ratio, proportion, interest, time and work. Having a strong hold on these concepts can help you understand the concept of work efficiency and how interest is accrued on bank savings. It can also guide you in time management, work planning,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

### Personality

## Competencies



### Extraversion

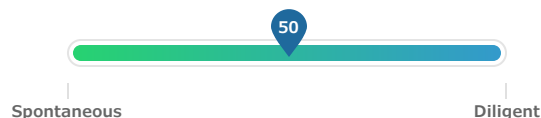


Extraversion refers to a person's inclination to prefer social interaction over spending time alone. Individuals with high levels of extraversion are perceived to be outgoing, warm and socially confident.

- You are 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

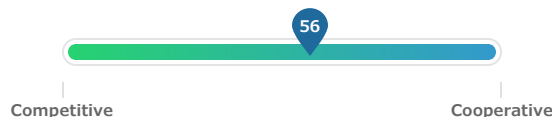


Conscientiousness is the tendency to be organized, hard working and responsible in one's approach to your work. Individuals with high levels of this personality trait are more likely to be ambitious and tend to be goal-oriented and focused.

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



### Agreeableness



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

- You are flexible regarding your opinions and be willing to accommodate the needs of others.
- You are generally considerate of the needs of others yet may, at times, overlook social norms to achieve personal success.
- You are selective about the people you choose to trust.
- You are caring and you empathise a friend in distress.
- You give credit to others but also tends to be open with your friends about personal achievements.

- You are more inclined to strike a compromise in tough situations and may be suitable for jobs that demand managing expectations among different stakeholders.



## Openness to Experience

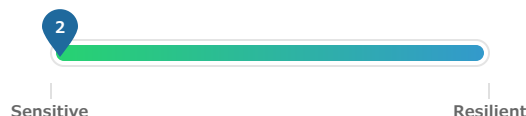


Openness to experience refers to a person's inclination to explore beyond conventional boundaries in different aspects of life. Individuals with high levels of this personality trait tend to be more curious, creative and innovative in nature.

- You 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 pursue multiple tasks simultaneously, switching between them when needed.
- You prefer working to achieve some progress on multiple tasks simultaneously than completing one task before moving on to the next task.
- You tend to believe that multitasking is an efficient way of doing things and prefers an action packed work life with multiple projects.





### 3 | Response

#### WriteX - Essay Writing



83 / 100

CEFR: C1

#### Question


The use of messaging apps like WhatsApp/WeChat have bridged the communication gap between people living far apart.


Do you agree or disagree? In your view, how have these apps transformed relationships? Substantiate your response with reasons and suitable examples.

#### Scores

Content Score

Grammar Score






 86 / 100

 78 / 100

#### Response

I strongly agree that messaging apps like WhatsApp and WeChat have bridged the communication gap between people living far apart, transforming relationships in profound ways. These platforms have revolutionized how individuals stay connected, making long-distance communication not only possible but also more immediate and personal. Prior to such apps, maintaining relationships over long distances often involved expensive phone calls or delayed letters, which made staying in touch difficult, especially for those in remote locations. Messaging apps have significantly lowered these barriers by allowing users to send instant text messages, voice notes, images and even video calls at little to no cost, overcoming geographical constraints. WhatsApp, for instance, has become an indispensable tool for personal, family, and professional communication worldwide. It allows users to communicate in real-time, share multimedia, and organise group chats, all from the convenience of their smartphones. WeChat, similarly, offers diverse features like mobile payments, integrated social media, and messaging, providing a more immersive and multifunctional platform. These capabilities not only make conversations more dynamic but also provide an easy way for people to share their lives, from simple updates to significant milestones. These apps have also played a key role in strengthening relationships between friends, families, and even business partners. For example, families with members living in different countries can now share moments through photos or video calls, maintaining emotional bonds despite the distance. In business, teams across continents can collaborate seamlessly through group chats and file sharing, enhancing productivity and communication efficiency. Furthermore, apps like WhatsApp have proven to be vital in emergency situations, allowing people to quickly alert their loved ones or seek help when needed. In conclusion, messaging apps like WhatsApp and WeChat not only make communication more efficient but have also transformed the dynamics of relationships. They have provided a platform that fosters intimacy, strengthens bonds, and facilitates immediate interaction, thereby bridging the distance between people and fostering global connectedness.

#### Error Summary

	Spelling	7
	White Space	0
	Style	0
	Grammar	7
	Typographical	0

## Essay Statistics

317

Total words

13

Total sentences

24

Average sentence  
length

215

Total unique words

96

Total stop words

## Error Details

### Spelling

... in profound ways. These platforms have revolutionized how individuals stay connected, making long-distance communication possible.

Possible spelling mistake found

... These platforms have revolutionized how individuals stay connected, making long-distance communication possible.

Possible spelling mistake found

...pensible tool for personal, family, and professional communication worldwide. It allows users to communicate in real-time.

Possible spelling mistake found

... family, and professional communication worldwide. It allows users to communicate in real-time.

Possible spelling mistake found

...gathering relationships between friends, families, and even business partners. For example, families with members living in different locations can stay connected through messaging apps.

Possible spelling mistake found

...een friends, families, and even business partners. For example, families with members living in different locations can stay connected through messaging apps.

Possible spelling mistake found

...oved ones or seek help when needed. In conclusion, messaging apps like WhatsApp and WeChat have revolutionized communication.

Possible spelling mistake found

### Grammar

Prior to such apps, maintaining relationships over long distances often involved expensive phone calls or delayed letters, which made staying in touch difficult, especially for those in remote locations. Messaging apps have significantly lowered these barriers by allowing users to send instant text messages, voice notes, images and even video calls at little to no cost, overcoming geographical constraints.

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

Prior to such apps, maintaining relationships over long distances often involved expensive phone calls or delayed letters, which made staying in touch difficult, especially for those in remote locations. Messaging apps have significantly lowered these barriers by allowing users to send instant text messages, voice notes, images and even video calls at little to no cost, overcoming geographical constraints.

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

Prior to such apps, maintaining relationships over long distances often involved expensive phone calls or delayed letters, which made staying in touch difficult, especially for those in remote locations. Messaging apps have significantly lowered these barriers by allowing users to send instant text messages, voice notes, images and even video calls at little to no cost, overcoming geographical constraints.

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

WhatsApp, for instance, has become an indispensable tool for personal, family, and professional **communication** word wide.

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

It allows users to **communicated** in real-time, share multimedia, and organise group chats, all from the convenience of their smartphones.

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

It allows users to communicated in real-time, share multimedia, and organise group **chats**, all from the convenience of their smartphones.

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

WeChat, similarly, offers diverse features like mobile payments, integrated social **media**, and messaging, providing a more immersive and multifunctional platform.

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

## Automata Pro



96 / 100

[Code Replay](#)

### Question 1 (Language: Python)

An employee has to send a secret code  $S$  to their boss. They design a method to encrypt the code using two key values  $N$  and  $M$ . The formula that they use to develop the encrypted code is shown below:

$$(((S^N \% 10)^M) \% 1000000007)$$

Write an algorithm to help the employee encrypt the code.

### Scores

#### Programming Ability



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

#### Functional Correctness



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

#### Programming Practices



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

### Final Code Submitted

Compilation Status: Pass

```
1
2 """
3
4 """
5 def findSecretCode(secretCode, firstKey, secondKey):
6     last_digit = pow(secretCode, firstKey, 10)
```

### Code Analysis

#### Average-case Time Complexity

Candidate code:  $O(\log N)$

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

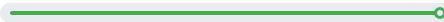
```

7 encrypted_code = pow(last_digit, secondKey, 1000000007)
8 return encrypted_code
9
10 def main():
11     # input for secretCode
12     secretCode = int(raw_input())
13     # input for firstKey
14     firstKey = int(raw_input())
15     # input for secondKey
16     secondKey = int(raw_input())
17
18     result = findSecretCode(secretCode, firstKey, secondKey)
19     print result
20
21 if __name__ == "__main__":
22     main()

```

\*N represents key value of N/M

<b>Errors/Warnings</b>
There are no errors in the candidate's code.
<b>Structural Vulnerabilites and Errors</b>
There are no errors in the candidate's code.

<b>Test Case Execution</b>		Passed TC: 100%		
Total score  17/17		<b>100%</b> Basic(8/8)	<b>100%</b> Advance(5/5)	<b>100%</b> Edge(4/4)

<b>Compilation Statistics</b>					
<div>4</div> Total attempts	<div>3</div> Successful	<div>1</div> Compilation errors	<div>0</div> Sample failed	<div>2</div> Timed out	<div>0</div> Runtime errors
Response time:			00:11:46		
Average time taken between two compile attempts:			00:02:57		
Average test case pass percentage per compile:			33.82%		

### 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: Python)


A virtual memory management system in an operating system uses First-In-First-Out (FIFO) cache.

When a requested memory page is not in the cache and the cache is full, the page that has been in the cache for the longest duration is removed to make room for the requested page. If the cache is not full, then the requested page can simply be added to the cache. A given page should occur once in the cache at most.

Given the maximum size of the cache and an array of page requests, calculate the number of cache misses. A cache miss occurs when a page is requested but is not found in the cache. Initially, the cache is empty.

### Scores

#### Programming Ability

 **80** / 100

Correct with inadvertent errors. Correct control structures and critical data dependencies incorporated. Some inadvertent errors make the code fail test cases.

#### 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.

#### Programming Practices

 **100** / 100

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

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 2 """ 3 page_requests, representing the array with size of page_requests_s   size. 4 max_cache_size, representing the size of the cache. 5 """ 6 from collections import deque 7 def cacheMisses(page_requests, max_cache_size): 8     cache = deque() 9     cache_misses = 0 10 11     for page in page_requests: 12         if page not in cache: 13             cache_misses += 1 14             if len(cache) &gt;= max_cache_size: 15                 cache.popleft() 16                 cache.append(page) 17     return cache_misses 18 19 def main(): 20     # input for page_requests 21     page_requests = [] 22     page_requests_size = int(raw_input()) 23     page_requests = list(map(int,raw_input().split())) 24     # input for max_cache_size 25     max_cache_size = int(raw_input()) 26 27     result = cacheMisses(page_requests, max_cache_size) 28     print (result) 29 30 if __name__ == "__main__": 31     main() </pre>		<p><b>Average-case Time Complexity</b></p> <p><b>Candidate code:</b> Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p><b>Best case code:</b> <math>O(N)</math></p> <p>*N represents number of requested page references</p>
		<p><b>Errors/Warnings</b></p> <p>There are no errors in the candidate's code.</p>
		<p><b>Structural Vulnerabilites and Errors</b></p> <p>There are no errors in the candidate's code.</p>

Test Case Execution		Passed TC: 88.89%		
<p>Total score</p> <div> <div></div> <div>16/18</div> </div>		<p><b>83%</b></p> <p>Basic(5/6)</p>	<p><b>91%</b></p> <p>Advance(10/11)</p>	<p><b>100%</b></p> <p>Edge(1/1)</p>

## Compilation Statistics

2

Total attempts

2

Successful

0

Compilation errors

0

Sample failed

0

Timed out

1

Runtime errors

Response time:

00:06:05

Average time taken between two compile attempts:

00:03:03

Average test case pass percentage per compile:

50%

### Average-case Time Complexity

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

### Test Case Execution

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

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

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

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

## Automata Fix



43 / 100

[Code Replay](#)

### Question 1 (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     int mid = size/2;
5     for(i=0; i< (size-mid)/2 ; i++)
6     {
7         temp = inputList[size-i-1];
8         inputList[size-i-1] = inputList[mid + i];
9         inputList[mid + i] = temp;
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.

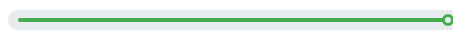
#### Structural Vulnerabilities and Errors

There are no errors in the candidate's code.

### Test Case Execution

Passed TC: 100%

Total score

 8/8

**100%**

Basic(4/4)

**100%**

Advance(2/2)

**100%**

Edge(2/2)

### Compilation Statistics

4

Total attempts

4

Successful

0

Compilation errors

3

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:05:09

Average time taken between two compile attempts:

00:01:17

Average test case pass percentage per compile:

25%



## 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++)

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 using namespace std;
3 int difference_in_times(Time *time1, Time *time2)
4 {
5     // write your code here
6 }
7
```

### 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\_24.cpp:8:  
source\_24.cpp: In function 'int difference\_in\_times(Time\*, Time\*)':  
source\_24.cpp:6:1: error: no return statement in function returning non-void [-Werror=return-type]  
}

^  
cc1plus: some warnings being treated as errors

#### Structural Vulnerabilities and Errors

There are no errors in the candidate's code.

### Compilation Statistics

1

Total attempts

0

Successful

1

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:10

Average time taken between two compile attempts:

00:00:10

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 3 (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	Code Analysis
<pre> 1 #include &lt;iostream&gt; 2 using namespace std; 3 void manchester(int size, int* arr) 4 { 5     int count =0; 6     int* res = new int[size]; 7     for(int i = 0; i &lt; size; i++) 8     { 9         if(i==0) 10             res[i]= 0; 11         else if (arr[i] == arr[i-1]) 12             { count++; 13             res[i]=count; } 14         else { 15             count = 0; 16             res[i] = count; 17         } 18     } 19 } 20 for(int i=0; i&lt;size; i++) 21 { 22     cout&lt;&lt;res[i]; 23     if(i!=size-1) 24         cout &lt;&lt; " "; 25 } 26 cout &lt;&lt; endl; 27 28 delete[] res; 29 }</pre>		<p><b>Average-case Time Complexity</b></p> <p><b>Candidate code:</b> Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p><b>Best case code:</b></p> <p>*N represents</p>
		<p><b>Errors/Warnings</b></p> <p>There are no errors in the candidate's code.</p>
		<p><b>Structural Vulnerabilites and Errors</b></p> <p>There are no errors in the candidate's code.</p>

## Test Case Execution

Passed TC: 0%

Total score

0/7

0%

Basic(0/5)

0%

Advance(0/1)

0%

Edge(0/1)

## Compilation Statistics

7

Total attempts

7

Successful

0

Compilation errors

7

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:07:22

Average time taken between two compile attempts:

00:01:03

Average test case pass percentage per compile:

0%

### *i* Average-case Time Complexity

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

### *i* Test Case Execution

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

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

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

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

## Question 4 (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 ***isTriangle*** which accepts three points *P1*, *P2*, *P3* as inputs and checks whether the given three points form a triangle.

If they form a triangle, the function/method returns an integer 1. Otherwise, it returns an integer 0.

.

### 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 using namespace std;
3 int isTriangle(Point *P1, Point *P2, Point *P3)
4 {
5     // write your code here
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.

**Best case code:**

\*N represents

#### Errors/Warnings

In file included from main\_25.cpp:8:  
 source\_25.cpp: In function 'int isTriangle(Point\*, Point\*, Point\*)':  
 source\_25.cpp:6:1: 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

Total attempts

0

Successful

1

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:00:16

Average time taken between two compile attempts:

00:00:16

Average test case pass percentage per compile:

0%

## *i* Average-case Time Complexity

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

## *i* Test Case Execution

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

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

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

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

## Question 5 (Language: C++)

The function/method **removeKLargestElement** prints space-separated integers that are left after removing the  $K^{\text{th}}$  largest integer from the input list *arr*. If the given *K* is out of bounds ( $K > \text{len}$ ) or  $K^{\text{th}}$  largest integer does not exist, then the function/method should print space-separated integers of the input list *arr*.

The function/method **removeKLargestElement** accepts three arguments - *len*, an integer representing the number of elements in the list, *K*, an integer value and *arr*, a list of integers.

The function/method **removeKLargestElement** uses another error-free function/method **kLargestElement** which accepts three arguments - *len*, an integer representing the number of elements in the list, *K*, an integer value and *arr*, a list of integers and returns an integer representing the  $K^{\text{th}}$  largest integer in the input list *arr*.

The function/method **removeKLargestElement** compiles unsuccessfully due to syntactical error. Your task is to

debug the code so that it passes all the test cases.

### Assumptions:

The inputs *len* and *K* are always non-negative integers.

### Scores

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 int kLargestElement(int len, int K, vector&lt;int&gt; arr) 2 { 3     int kth,temp; 4     int *rarr = (int*)malloc(sizeof(int)*(len)); 5     for(int i=0;i&lt;len;i++) 6         rarr[i]= arr[i]; 7 8     for(int i1 = 0; i1&lt;len; i1++ ) 9     { 10         for(int j1 = i1+1; j1&lt;len; j1++) 11         { 12             if(rarr[i1]&gt;rarr[j1]) 13             { 14                 temp = rarr[i1]; 15                 rarr[i1] = rarr[j1]; 16                 rarr[j1] = temp; 17             } 18         } 19     } 20     kth=rarr[len-K]; 21     return kth; 22 } 23 24 void removeKLargestElement(int len, int K, vector&lt;int&gt; arr) 25 { 26     vector&lt;int&gt; rarr1; 27     if(K&gt;0 &amp;&amp; K &lt;= len) 28     { 29         int i, kth; 30         kth=kLargestElement(len, K, arr); 31         int pos=0; 32         for(int j1=0;j1&lt;len;j1++) 33         { 34             if(arr[j1]==kth) 35             { 36                 pos=j1; 37             } 38         } 39         for(i=pos;i&lt;len-1;i++) </pre>		<b>Average-case Time Complexity</b> <p><b>Candidate code:</b> Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p><b>Best case code:</b></p> <p>*N represents</p>
		<b>Errors/Warnings</b> <p>There are no errors in the candidate's code.</p>
		<b>Structural Vulnerabilites and Errors</b> <p>There are no errors in the candidate's code.</p>



```

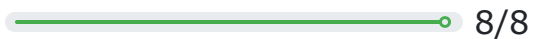
40  {
41      arr[i]=arr[i+1];
42  }
43  for(int i=0;i<len-1;i++)
44  {
45      cout<<arr[i];
46      if(i<len-2)
47      {
48          cout<<" ";
49      }
50  }
51  }
52  else
53  {
54      for(int i=0;i<len;i++)
55      {
56          cout<<arr[i];
57          if(i<len-1)
58          {
59              cout<<" ";
60          }
61      }
62  }
63 }

```

### Test Case Execution

Passed TC: 100%

Total score



**100%**

Basic(8/8)

**0%**

Advance(0/0)

**0%**

Edge(0/0)

### Compilation Statistics

2

Total attempts

1

Successful

1

Compilation errors

0

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:01:57

Average time taken between two compile attempts:

00:00:59

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 6 (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

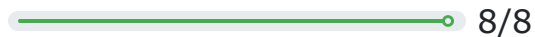
Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 #include &lt;iostream&gt; 2 using namespace std; 3 void printFibonacci(int num) 4 { 5     long num1 = 0; 6     long num2 = 1; 7     for (int i = 1; i &lt;= num; ++i) 8     { 9         cout&lt;&lt;num1&lt;&lt;" "; 10        long sum = num1 + num2; 11        num1 = num2; 12        num2 = sum; 13    } 14    cout &lt;&lt; endl; 15 }</pre>		<b>Average-case Time Complexity</b> <p><b>Candidate code:</b> Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p><b>Best case code:</b></p> <p>*N represents</p>
		<b>Errors/Warnings</b> <p>There are no errors in the candidate's code.</p>
		<b>Structural Vulnerabilities and Errors</b>

There are no errors in the candidate's code.

### Test Case Execution

Passed TC: 100%

Total score



100%

Basic(5/5)

100%

Advance(2/2)

100%

Edge(1/1)

### Compilation Statistics

7

Total attempts

7

Successful

0

Compilation errors

6

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:02:57

Average time taken between two compile attempts:

00:00:25

Average test case pass percentage per compile:

14.3%

### *i* Average-case Time Complexity

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

### *i* Test Case Execution

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

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

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

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

### Question 7 (Language: 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.

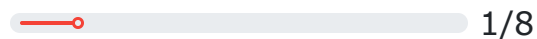
## Scores

Final Code Submitted	Compilation Status: Pass	Code Analysis
<pre> 1 #include &lt;iostream&gt; 2 using namespace std; 3 void printPattern(int num) 4 { 5     int i,j; 6     for(i=1;i&lt;=num;i++) 7     { 8         for(j=i;j&lt;=2*i;j++) 9         { 10             cout&lt;&lt;j&lt;&lt;" "; 11         } 12         cout&lt;&lt;"\n"; 13     } 14 } 15 </pre>		<b>Average-case Time Complexity</b> <p><b>Candidate code:</b> Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.</p> <p><b>Best case code:</b></p> <p>*N represents</p>
		<b>Errors/Warnings</b> <p>There are no errors in the candidate's code.</p>
		<b>Structural Vulnerabilites and Errors</b> <p>There are no errors in the candidate's code.</p>

## Test Case Execution

Passed TC: 12.5%

Total score



0%

Basic(0/7)

0%

Advance(0/0)

100%

Edge(1/1)

## Compilation Statistics

3

Total attempts

3

Successful

0

Compilation errors

3

Sample failed

0

Timed out

0

Runtime errors

Response time:

00:01:27

Average time taken between two compile attempts:

00:00:29

Average test case pass percentage per compile:

0%

### Average-case Time Complexity

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

### Test Case Execution

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

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

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

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

## 4 | Learning Resources

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