Gitashri Sathyamoorthy

Test ID: 450011103000076 | 8072967157 | gitasathya2407@gmail.com

Test Date: April 6, 2025

Computer Science 100 /100	Logical Ability 64 /100	Data Science 58 /100	Computer Programming 79 /100
Quantitative Ability (Advanced) 74 /100	English Comprehension 66 /100	WriteX - Essay Writing 82 /100	Automata 64 /100
Automata Fix 86 /100	Personality ✓ Completed		

Computer Science

100 / 100

OS and Computer Architecture	DBMS	Computer Networks
100 / 100	67 / 100	100 / 100

Logical Ability

64 / 100

Inductive Reasoning	Deductive Reasoning	Abductive Reasoning
60 / 100	71 / 100	60 / 100

Data Science

58 / 100

ML Algorithms and Implementation	Probability and Statistics	ML Experiments
60 / 100	50 / 100	67 / 100

Computer Programming

79 / 100

Basic Programming	Data Structures	OOP and Complexity Theory
84 / 100	83 / 100	71 / 100

Quantitative Ability (Advanced)

74 / 100

Basic Mathematics	Advanced Mathematics	Applied Mathematics
67 / 100	74 / 100	82 / 100

English Comprehension

66 / 100 | CEFR: **C1**

Grammar	Vocabulary	Comprehension
69 / 100	68 / 100	61 / 100

WriteX - Essay Writing

82 / 100 | CEFR: **C1**

Content Score	Grammar Score
82 / 100	82 / 100

Automata

64 / 100

Programming Ability	Programming Practices	Functional Correctness
60 / 100	100 / 100	60 / 100

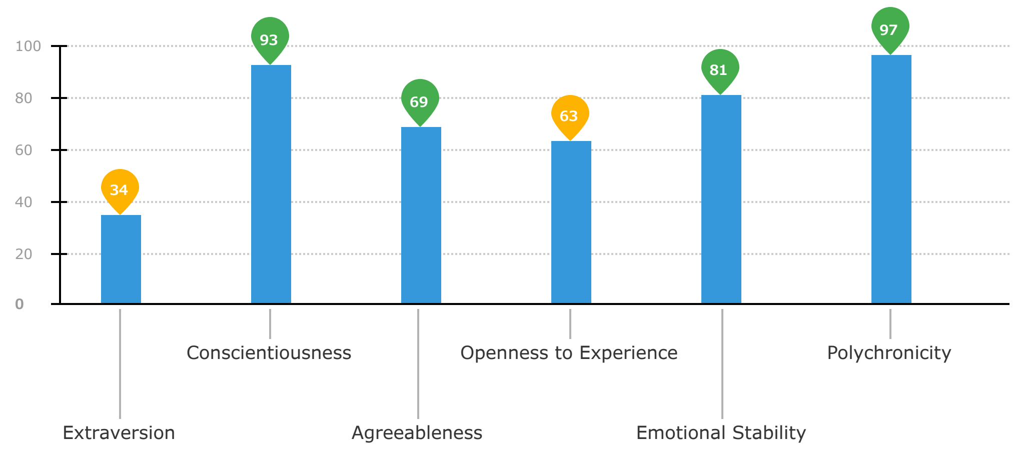
Automata Fix

86 / 100

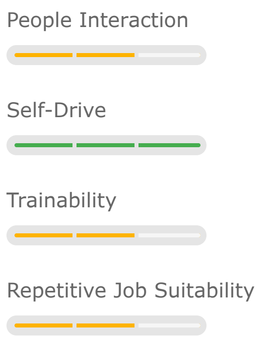
Logical Error	Code Reuse	Syntactical Error
75 / 100	100 / 100	100 / 100

Personality

Completed



Competencies



Work attributes

1 | Introduction

About the Report

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

This report has the following sections:

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

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

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

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

Score Interpretation

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

- 🟢 Scores between 67 and 100
- 🟡 Scores between 33 and 67
- 🔴 Scores between 0 and 33

2 | Insights

English Comprehension

66 / 100 | CEFR: **C1**

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

64 / 100

Inductive Reasoning

60 / 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

71 / 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

60 / 100

Quantitative Ability (Advanced)

74 / 100

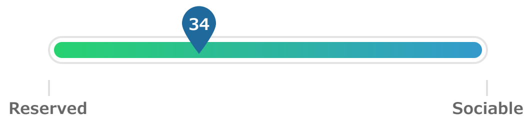
This test aims to measure your ability to solve problems on basic arithmetic operations, probability, permutations and combinations, and other advanced concepts.

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

Personality

Competencies

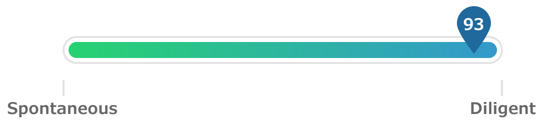
Extraversion



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

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

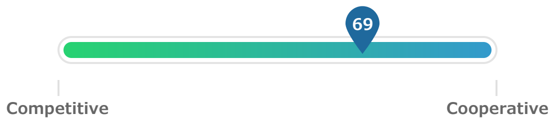
Conscientiousness



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

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

Agreeableness

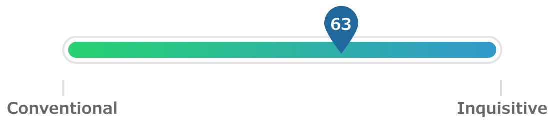


Agreeableness refers to an individual's tendency to be cooperative with others and it defines your approach to interpersonal relationships. People with high levels of this personality trait tend to be more considerate of people around them and are more likely to work effectively in a team.

- You are considerate and sensitive to the needs of others.
- You tend to put the needs of others ahead of your own.
- You are likely to trust others easily without doubting their intentions.
- You are compassionate and may be strongly affected by the plight of both friends and strangers.
- You are humble and modest and prefer not to talk about personal accomplishments.

- Your personality is more suitable for jobs demanding cooperation among employees.

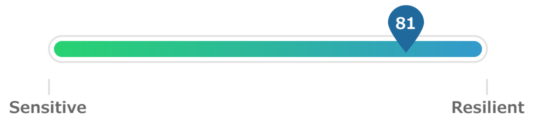
Openness to Experience



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

- You 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.

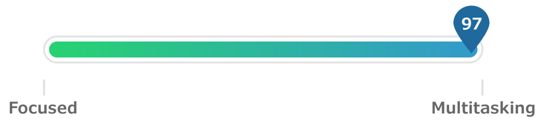
Emotional Stability



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 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

82 / 100 | CEFR: **C1**

Question

Some parents feel that sports is a distraction to their kids' studies. There are others who give due importance to sports for the holistic development of a child.
What is your view? Support your response with reasons and examples.

Scores

Content Score	Grammar Score
82 / 100	82 / 100

Response

Greetings, I'm Gitashri Sathyamoorthy. On this topic, it's better to understand the indi vidual kid's development rather than decide on the parents' POV. It's important to unde rstand the children's minds. Some parents believe that sports distract their Child's acad emic success. however, I strongly believe that each parent should analyse their childre n's mindset and should decide upon their future. Sports play a vital role in the holistic development of a child and should be given due importance alongside studies. Engagin g in sports helps children develop physical fitness, discipline, and time management ski lls. It teaches them valuable life lessons such as teamwork, leadership, and handling b oth victory and defeat gracefully. These experiences contribute significantly to shaping their character and personality. Moreover, regular Physical activity has been scientificall y proven to improve concentration, memory, and overall mental well-being. A child wh o participates in sports is likely to be more focused and energetic during academic acti vities. for example, students who engage in sports often show better academic perform ance due to improved brain functions and stress relief. in today's competitive world, su ccess is not only defined by academic excellence but also by the ability to handle press ure, work in teams, and maintain good health - all of which can be achieved through sp orts. Therefore, instead of viewing sports as a distraction, parents should encourage th eir children to maintain a balanced lifestyle that includes both studies and physical acti vities. Also, when we see today's young prodigies, every kid is talented in every aspec t, Painting, Designing, etc. As a parent, it's their job to notice these special talents in t heir kid and to assist them in achieving greatness in life rather than choosing the field f or them. In the olden days of India, all students would choose either the Engineering fi eld or the Doctor field because their parents forced them to choose between these two. Studying Arts was not at all an option for students who were genuinely interested in it. However, nowadays, the situation has changed a lot, and people understand their kids' feelings, because they have been through it. In conclusion, I would like to say that all o f us should be assisting our children in choosing their career with joy and not by forcin g them like our parents did to us. and also our future motherlands' mobility and aspect s of living lie on their shoulders. Thank you for giving the opportunity to use my voice. Weiogether

Error Summary

Spelling	3
White Space	3
Style	0
Grammar	17
Typographical	4

Essay Statistics

400	23	17	245	162
Total words	Total sentences	Average sentence length	Total unique words	Total stop words

Error Details

Spelling

Greetings, I'm Gitashri Sathyamoorthy. On this topic, it's b ett...	Possible spelling mistake found
Greetings, I'm Gitashri Sathyamoorthy. On this topic, it's b etter to understa...	Possible spelling mistake found
...giving the opportunity to use my voice. Weiogether	Possible spelling mistake found

White Space

...s both studies and physical activities. Also, when we see today's young prodigie...	Possible typo: you repeated a whitespace
...ts who were genuinely interested in it. However, nowada ys, the situation has cha...	Possible typo: you repeated a whitespace
...pects of living lie on their shoulders. Thank you for givin g the opportunity to ...	Possible typo: you repeated a whitespace

Grammar

On this topic, it's better to understand the individual kid's development rather than decide on the parents' POV.	Possible grammar error found. Consider inserting "it" over here.
It's important to understand the children's minds.	Possible grammar error found. Consider removing "the" from here.
It's important to understand the children's minds.	Possible grammar error found. Consider replacing it with "children".
It's important to understand the children's minds.	Possible grammar error found. Consider inserting "is" over here.
however, I strongly believe that each parent should analys e their children's mindset and should decide upon their fut ure.	Possible grammar error found. Consider replacing it with "children".
however, I strongly believe that each parent should analys e their children's mindset and should decide upon their fut ure.	Possible grammar error found. Consider inserting "is" over here.
Engaging in sports helps children develop physical fitness, discipline, and time management skills.	Possible grammar error found. Consider replacing it with "discipline".

in today's competitive world, success is not only defined by academic excellence but also by the ability to handle press ure, work in teams, and maintain good health - all of which can be achieved through sports.	Possible grammar error found. Consider replacing it with "today".
in today's competitive world, success is not only defined by academic excellence but also by the ability to handle press ure, work in teams, and maintain good health - all of which can be achieved through sports.	Possible grammar error found. Consider inserting "is" over here.
in today's competitive world, success is not only defined by academic excellence but also by the ability to handle press ure, work in teams, and maintain good health - all of which can be achieved through sports.	Possible grammar error found. Consider replacing it with "teams".
Also, when we see today's young prodigies, every kid is tal ented in every aspect, Painting, Designing, etc.	Possible grammar error found. Consider replacing it with "today".
Also, when we see today's young prodigies, every kid is tal ented in every aspect, Painting, Designing, etc.	Possible grammar error found. Consider inserting "is" over here.
Also, when we see today's young prodigies, every kid is tal ented in every aspect, Painting, Designing, etc.	Possible grammar error found. Consider replacing it with "designing".
As a parent, it's their job to notice these special talents in t heir kid and to assist them in achieving greatness in life rat her than choosing the field for them.	Possible grammar error found. Consider inserting "it" over here.
As a parent, it's their job to notice these special talents in t heir kid and to assist them in achieving greatness in life rat her than choosing the field for them.	Possible grammar error found. Consider replacing it with "kids".
As a parent, it's their job to notice these special talents in t heir kid and to assist them in achieving greatness in life rat her than choosing the field for them.	Possible grammar error found. Consider replacing it with "help".
and also our future motherlands' mobility and aspects of li ving lie on their shoulders. Thank you for giving the opport unity to use my voice.	Possible grammar error found. Consider inserting "me" over here.

Typographical

...istract their Child's academic success. however, I strongl y believe that each parent sh...	This sentence does not start with an uppercase letter
...d energetic during academic activities. for example, stud ents who engage in sports ...	This sentence does not start with an uppercase letter
...ved brain functions and stress relief. in today's competiti ve world, success is n...	This sentence does not start with an uppercase letter
...work in teams, and maintain good health - all of which c an be achieved through sp...	Consider using an m-dash if you do not want to join two words.

Automata

64 / 100 Code Replay

Question 1 (Language: Python) High similarity detected : **100%**

A stock trader trades in N selected stocks. The trader has calculated the relative stock price changes in the N stocks from the previous day's stock prices. The trader's lucky number is K, so the trader wants to invest in the particular stock that has the K^{th} smallest relative stock value.

Write an algorithm for the trader to find the K^{th} smallest stock price among the selected N stocks.

Scores

Programming Ability

100 / 100

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

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

100 / 100

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

Final Code Submitted

Compilation Status: Pass

```

"""

"""
def smallestStockPrice(stock, valueK):
    # Write your code here
    stock.sort()


    return stock[valueK-1]

def main():
    # input for stock
    stock = []
    stock_size  = int(raw_input())
    stock = list(map(int,raw_input().split()))
    # input for valueK
    valueK = int(raw_input())

    result = smallestStockPrice(stock, valueK)
    print (result)

if __name__ == "__main__":
```

Code Analysis

Average-case Time Complexity

Candidate code: O(log N)

Best case code: O(log N)

*N represents number of selected stocks.

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

```
main()
```

Test Case Execution

Passed TC: **100%**

Total score 9/9

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

Compilation Statistics

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

Response time:	**00:04:40**
Average time taken between two compile attempts:	**00:02:20**
Average test case pass percentage per compile:	**100%**

Similarity Detected

Every response is checked against:

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

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

	Candidate Response		Peer's Response
4	def smallestStockPrice(stock, valueK):	4	def smallestStockPrice(stock, valueK):
5	# Write your code here	5	# Write your code here
6	stock.sort()	6	stock.sort()
7			
8			
9	return stock[valueK-1]	7	return stock[valueK-1]
10		8	
11	def main():	9	def main():
...		...	
17	valueK = int(raw_input())	15	valueK = int(raw_input())

18		16	
19	`    result = smallestStockPrice(stock, valueK)`	17	`    result = smallestStockPrice(stock, valueK)`
20	`    print (result)`	20	`    print (result)`
21		19	
22	`if __name__ == "__main__":`	20	`if __name__ == "__main__":`
23	`    main()`	21	`    main()`

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 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 Ability

20 / 100

Code seems to be unrelated to the given problem.

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

20 / 100

The source code does not pass any basic test cases. It is either due to incorrect logic or runtime errors. Some advanced or edge cases may randomly pass.

Final Code Submitted — Compilation Status: Pass

```
1
2  """
3  parentPos, is an integer representing the initial position of the child's parent.
4  childPos, is an integer representing the initial position of the child.
5  velParent, is an integer representing the velocity of the parent.
6  steps, is an integer representing the number of steps taken by the parent.
7  """
8  def commonFootsteps(parentPos, childPos, velParent, steps):
9      #parent_steps = set(parentPos+i * velParent for i in range(steps))
10     #farthest_parent_pos = parentPos +(steps -1) *velParent
11     # Write your code here
12     max_common = 0
13     best_velChild=0
14     parent_end  = parentPos+(steps-1)*velParent
15
16     for velChild in range(1,101):
17         delta = childPos-parentPos
18         g= math.gcd(velParent, velChild)
19
20         if delta%g != 0:
21             continue
22
23         lcm = (velParent + velChild)//g
24         first_common = ((max(childPos,parentPos)-childPos + velChild -1)//velChild) * velChild + childPos
25
26         if first_common>parent_end:
27             continue
28
29         count =(parent_end-first_common)//lcm + 1
```

Code Analysis

Average-case Time Complexity

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

Best case code: 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

Readability & Language Best Practices

Line 24: line too long (100 > 79 characters)
Line 13,16,18...: Variables/Arguments are not given proper names

Performance & Correctness

Line 18: undefined name 'math'

```python

        if count > max_common or (count == max_common and velChild>best_velChild):
            max_common = count
            best_velChild = velChild


    return max_common, best_velChild

def main():
    # input for parentPos
    parentPos = int(raw_input())
    # input for childPos
    childPos = int(raw_input())
    # input for velParent
    velParent = int(raw_input())
    # input for steps
    steps = int(raw_input())

    result = commonFootsteps(parentPos, childPos, velParent, steps)
    print(result[0], result[1])

if __name__ == "__main__":
    main()
```

Test Case Execution

Passed TC: **0%**

Total score 0/10

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

Compilation Statistics

Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors
5	5	0	0	3	2

Response time:	**00:26:50**
Average time taken between two compile attempts:	**00:05:22**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Automata Fix

● **86** / 100 | Code Replay

Question 1 (Language: Java)

Lisa always forgets her birthday which is on the 5th of July. So, develop a function/method which will be helpful to remember her birthday.

The function/method ***checkBirthDay*** return an integer '1' if it is her birthday else returns 0. The function/method ***checkBirthDay*** accepts two arguments - *month*, a string representing the month of her birthday and *day*, an integer representing the date of her birthday.

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

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
  int checkBirthDay(String month, int day)
  {
    if(month.equals("July") && (day==5))
      return 1;
    else
      return 0;
```

Code Analysis

Average-case Time Complexity

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

Best case code:

*N represents

```
  }
}
```

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score

9/9

100%	100%	0%
Basic(**6**/6)	Advance(**3**/3)	Edge(**0**/0)

Compilation Statistics

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

Response time:	**00:02:03**
Average time taken between two compile attempts:	**00:01:02**
Average test case pass percentage per compile:	**50%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 2 (Language: Java)

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

```
public class Point

{

   public int x, y;

   public Point()

   { }

   public Point(int x1, int y1)

   {

      x = x1 ;

      y = y1 ;

   }

   public double calculateDistance(Point P)

   {

      /*Return the euclidean distance between two input points.

        This can be called as -

        *  If P1 and P2 are two points then -

        *  P1.calculateDistance(P2);*/

   }

}
```

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
  int isTriangle(Point P1, Point P2, Point P3)
  {
    // write your code here
    int ares = P1.x * (P2.y - P3.y) + P2.x * (P3.y - P1.y) + P3.x * (P1.y -
P2.y);

    if(ares==0){
      return 0;
    }
    else{
      return 1;
    }
  }

}
```

Code Analysis

Average-case Time Complexity

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

Best case code:

*N represents

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution Passed TC: **100%**

Total score 9/9

100%	**100%**	**100%**
Basic(**6**/6)	Advance(**2**/2)	Edge(**1**/1)

Compilation Statistics

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

Response time:	**00:03:59**
Average time taken between two compile attempts:	**00:01:20**
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: Java)

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

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

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

Helper Description

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

```
public class Exponent

{

   public int base;

   public int exponent;

   int positiveExponent()

   {

       /*It calculate the Exponent for positive value of exponentValue
```

```
        This can be called as -

        Exponent exp = new Exponent(baseValue, exponentValue);

        float res = exp.positiveExponent();*/

    }

}
```

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
    float allExponent(int baseValue, int exponentValue)
    {
        float res = 1;
        if(exponentValue >=0)
        {
            Exponent exp = new Exponent(baseValue, exponentValue);
            res = (float)exp.positiveExponent();
        }
        else
        {
            // write your code here for negative exponentInput
            Exponent exp = new Exponent(baseValue, -exponentValue);
            res = 1.0f/exp.positiveExponent();
        }
        return res;
    }
}
```

Code Analysis

Average-case Time Complexity

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

Best case code:

*N represents

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score 6/6

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

Compilation Statistics

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

Response time:	**00:02:02**
Average time taken between two compile attempts:	**00:02:02**
Average test case pass percentage per compile:	**100%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 4 (Language: Java)

The function/method ***selectionSortArray*** performs an in-place selection sort on the given input list which will be sorted in ascending order.
The function/method ***selectionSortArray*** accepts two arguments - *len,* an integer representing the length of the input list and *arr*, a list of integers representing the input list, respectively*.*

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

Note:
In this particular implementation of selection sort, the smallest element in the list is swapped with the element at first index, the next smallest element is swapped with the element at the next index and so on.

Scores

Final Code Submitted — Compilation Status: Pass

```
// You can print the values to stdout for debugging
class Solution
{
  public void selectionSortArray( int len, int[] arr )
  {
    int x = 0 , y = 0 ;
    for( x = 0 ; x < len ; x ++ )
    {
      int index_of_min = x;
      for( y = x ; y < len ; y ++ )
      {
        if(arr[index_of_min]>arr[y])
        {
          index_of_min = y;
        }
      }
      int temp = arr[x];
      arr[x] = arr[index_of_min];
      arr[index_of_min] = temp;
    }
  }
}
```

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 9/9

100%	**100%**	**0%**
Basic(**4**/4)	Advance(**5**/5)	Edge(**0**/0)

Compilation Statistics

1	1	0	0	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:	**00:00:59**
Average test case pass percentage per compile:	**100%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 5 (Language: Java)

The function/method ***matrixSum*** returns an integer representing the sum of elements of the input matrix. The function/method ***matrixSum*** accepts three arguments - *rows*, an integer representing the number of rows of the input matrix, *columns*, an integer representing the number of columns of the input matrix and *matrix*, a two-dimensional array representing the input matrix.

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

Scores

Final Code Submitted **Compilation Status: Pass**

```java
// You can print the values to stdout for debugging
class Solution
{
 int matrixSum(int rows, int columns, int[][] matrix)
 {
   int i, j, sum=0;
   for(i=0;i<rows;i++)
   {
     for(j=0;j<columns;j++)
       sum += matrix[i][j];
   }
   return sum;
 }
}
```

Code Analysis

Average-case Time Complexity

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

Best case code:

*N represents

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score 10/10

100%	**100%**	**0%**
Basic(**6**/6)	Advance(**4**/4)	Edge(**0**/0)

Compilation Statistics

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

Response time:	**00:01:30**
Average time taken between two compile attempts:	**00:00:45**
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: Java)

The function/method ***patternPrint*** accepts an argument *num*, an integer.
The function/method ***patternPrint*** prints *num* lines in the following pattern.

For example, *num* = 4, the pattern should be:

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

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

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
class Solution
{
  void patternPrint(int num)
  {
    int print = 1;
    for(int i=1;i<=num;i++)
    {
      for(int j=1;j<=i;j++);
      {
        System.out.print("1");
      }
      System.out.println();
    }
  }
}

```

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: **25%**

Total score 2/8

14% Basic(**1**/7)

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

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

Compilation Statistics

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

Response time:	**00:03:36**
Average time taken between two compile attempts:	**00:00:54**
Average test case pass percentage per compile:	**9.4%**

Average-case Time Complexity

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

Test Case Execution

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

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

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

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

Question 7 (Language: Java)

The function/method ***descendingSortArray*** performs an in-place sort on the given input list which will be sorted in descending order.
The function/method ***descendingSortArray*** accepts two arguments - *len,* an integer representing the length of the input list and *arr*, a list of integers representing the input list, respectively*.*
5
The function/method ***descendingSortArray*** compiles successfully but fails to get the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Scores

Final Code Submitted **Compilation Status: Pass**

```
// You can print the values to stdout for debugging
```

Code Analysis

```
class Solution
{
    public void descendingSortArray(int len, int[] arr)
    {
        int small, pos, i, j, temp;
        for( i = 0 ; i <= len - 1 ; i ++ )
        {
            for( j = i ; j < len ; j ++ )
            {
                temp = 0;
                if( arr[i] < arr[j] )
                {
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }
    }
}
```

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%**	**0%**
Basic(**6**/6)	Advance(**4**/4)	Edge(**0**/0)

Compilation Statistics

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

Response time:	**00:00:43**
Average time taken between two compile attempts:	**00:00:43**
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 Shakespearean plays	🏷️	🌐	📖
Learn about how to get better at reading	💲	🌐	📖
Read opinions to improve your comprehension	🏷️	🌐	📖

Logical Ability

Learn about advanced deductive logic	🏷️	🌐	📖
Watch a video on the art of deduction	🏷️	🌐	🎬
Learn about Sherlock Holmes' puzzles and develop your deductive logic	🏷️	🌐	📖

Quantitative Ability (Advanced)

Learn about permutations and combinations: from the shallows to the deep	💲	🌐	📖
Learn about basic math for adults	💲	🌐	📖
Watch a video on how to calculate your Income Tax	🏷️	🌐	🎬

Icon Index

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