CHAPTER 7:

Computational Thinking

SOLVED EXERCISE

Tick (✓) the correct answer.

1. Which of the following best defines computational thinking?

a) A method of solving problems using mathematical calculations only.
b) A problem-solving approach that employs systematic, algorithmic, and logical thinking.
c) Atechniqueused exclusively in computer programming.
d) An approach that ignores real-world applications.

2. Why is problem decomposition important in computational thinking?

a) It simplifies problems by breaking them down into smaller, more manageable parts.
b) It complicates problems by adding more details.
c) It eliminates the need for solving the problem.
da) It is only useful for simple problems.

3. Pattern recognition involves:

a) Finding and using similarities within problems
b) Ignoring repetitive elements
c) Breaking problems into smaller pieces .
d) Writing detailed algorithms

4. Which term refers to the process of ignoring the details to focus on the main idea?

a) Decomposition b) Pattern recognition
c) Abstraction d) Algorithm design

5. Which of the following is a principle of computational thinking?

a) Ignoring problem understanding b) Problem simplification
c) Avoiding solution design d) Implementing random solutions

6. Algorithms are:

a) Lists of data
b) Graphical representations
c) Step-by-step instructions for solving a problem
d) Repetitive patterns

7. **Which of the following is the first step in problem-solving according to computational thinking?**
a) Writing the solution b) Understanding the problem
c) Designing a flowchart d) Selecting a solution

8. **Flowcharts are used to:**
a) Code a program b) Represent algorithms graphically
c) Solve mathematical equations d) Identify patterns

9. **Pseudocode is:**
a) A type of flowchart
b) A high-level description of an algorithm using plain language
c) A programming language d) A debugging tool

10. **Dry running a flowchart involves:**
a) Writing the code in a programming language
b) Testing the flowchart with sample data
c) Converting the flowchart into pseudocode
d) Ignoring the flowchart details

1	2	3	4	5	6	7	8	9	10
b	a	a	c	b	c	b	b	b	b

Short Answer Questions

1. Define computational thinking.
Ans: Computational Thinking (CT) is a problem-solving process that involves a set of skills and techniques to solve complex problems in a way that can be executed by a computer. This approach can be used in various fields such as computer science, biology, mathematics, and even daily life.

2. What is decomposition in computational thinking?
Ans: Decomposition is the method of breaking down a complicated problem into smaller, more convenient and manageable components. Let's take the example of building a birdhouse. This task might look tough at first, but if we break it down, we can handle each part one at a time.

3. Explain pattern recognition with an example.
Ans: Pattern recognition involves looking for similarities or patterns among and within problems. For example, if you notice that you always forget your homework on Mondays, you might recognize a pattern and set a reminder specifically for Sundays.

4. Describe abstraction and its importance in problem-solving.
Ans: Definition: Abstraction is the process of hiding the complex details while exposing only the necessary parts. It helps reduce complexity by allowing us to focus on the high-level overview without getting lost in the details.
Importance: Abstraction is a fundamental concept in problem solving, especially in computer science. This helps in understanding, designing, and solving problems more efficiently.

Example: Making a Cup of Tea - High-level Steps: 1. Boil water. 2. Add tea leaves or a tea bag. 3. Steep for a few minutes. 4. Pour into a cup and add milk/sugar if desired

5. What is an algorithm?

Ans: An algorithm is a step-by-step collection of instructions to solve a problem or complete a task.

Example: We imagine a recipe for baking a cake. The recipe provides a list of ingredients and step-by-step instructions to mix them and bake the cake. This is an example of an algorithm because it outlines a clear sequence of steps to achieve the goal of baking a cake

6. How does problem understanding help in computational thinking?

Ans: Understanding a problem involves identifying the core issue, defining the requirements, and setting the objectives. It is the first and most important step in problem-solving, especially in computational thinking**What are flowcharts and how are they used?**

Ans: Flowcharts are visual representations of the steps in a process using different symbols connected by arrows. They are widely used in various fields, including computer science, engineering, and business, to design systems, and communicate complex workflows clearly and effectively. It helps programmers and students focus on the logic of the algorithm without worrying about the syntax of a specific programming language.

7. Explain the purpose of pseudocode.

Ans: Pseudocode is a method of representing an algorithm using simple and informal language that is easy to understand. It combines the structure of programming clarity with the readability of plain English, making it a useful tool for planning and explaining algorithms.

8. How do you differentiate between flowcharts and pseudocode?

Ans:

Pseudocode	Flowcharts
· Pseudocode uses plain language and structured format to describe the steps of an algorithm. · It is read like a story, with each step is written out sequentially. · Pseudocode communicates the steps in a detailed, narrative -like format. · It is particularly useful for documenting algorithms and can be converted into actual code in any programming language.	· Flowcharts use graphical symbols and arrows to represent the flow of an algorithm. · It is like watching a movie, where each symbol (such as rectangles, diamonds, and ovals) represents a different type of action or decision, and arrows indicate the direction of the flow. · They are useful for identifying the steps and decisions in an algorithm at a glance.

9. What is a dry run and why is it important?

Ans: A dry run involves manually going through the algorithm with sample data to identify any errors. A dry run of a flowchart involves manually walking through the flowchart step-by-step to understand how the algorithm works without using a computer. This helps identify any logical errors and understand the flow of control.

10. Describe LARP and its significance in learning algorithms.

Ans: LARP stands for Logic of Algorithms for resolution of Problems. It is a fun and interactive way to learn how algorithms work by actually running them and seeing the results.

Significance:

LARP helps you to:

- Understand how algorithms work
- See the effect of different inputs on the output
- Practice writing and improving your own algorithms.

11. List and explain two debugging techniques.

Ans: Debugging is the process of finding and fixing errors in an algorithm or flowchart. Here are some common debugging techniques:

- Trace the Steps: Go through each step of your algorithm or flowchart to see identity where it goes wrong.
- Use Comments: Write comments or notes in your algorithm to explain what each part is supposed to do. This can help you spot mistakes.

Long Questions

1. Write an algorithm to assign a grade based on the marks obtained by a student. The grading system follows these criteria:

- 90 and above: A+
- 80 to 89: A
- 70 to 79: B
- 60 to 69: C
- Below 60: F

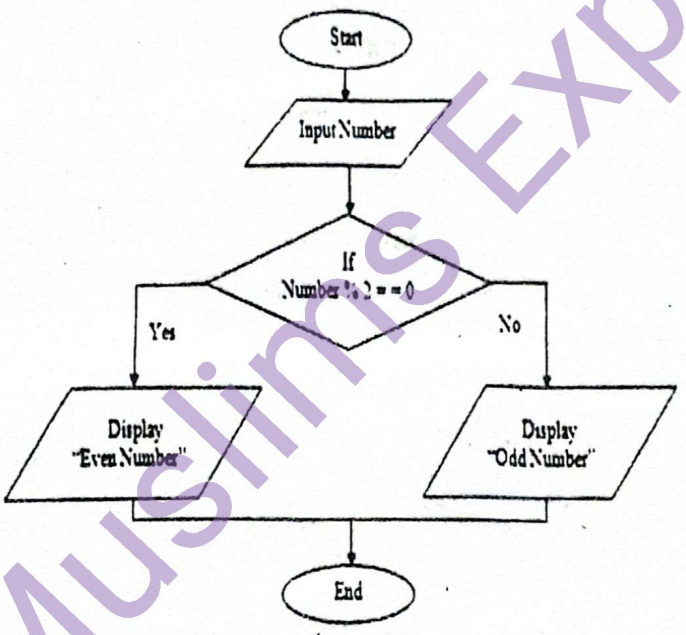
Ans: **Steps:**

1. Input marks obtained by the student
2. If marks are greater than or equal to 90, assign grade "A+"
3. Else if marks are greater than or equal to 80, assign grade "A"
4. Else if marks are greater than or equal to 70, assign grade "B"
5. Else if marks are greater than equal to 60, assign grade "C"
6. Else, assign grade "F".
7. Output: Display required grade

2. Explain how you would use algorithm design methods, such as flowcharts and pseudocode, to solve a complex computational problem. Illustrate your explanation with a detailed example.

Ans: Algorithm design methods provide a range of tools and techniques such as flowcharts and pseudocode to tackle various computational problems effectively. Each method has its strengths and weaknesses, making it suitable for different types of problems. Understanding these methods allows one to choose the most appropriate approach for a given problem, leading to more efficient and smart solutions.

Following flowchart will read a number from user. This number is checked using % operator to find whether it is odd or even.



Steps for the Flowchart

- **Start**: Begin the process.
- **Input the Number**: Accept a number from the user.
- **Check Condition**: Use the modulus operator to determine if the number is divisible by 2:
  - If the number % 2 equals 0, it is even.
  - Otherwise, it is odd.
- **Decision Box**:
  - If the condition (number % 2 == 0) is true, go to the "Even" path.
  - If the condition is false, go to the "Odd" path.
- **Output Result**: Display "Even" or "Odd" based on the path taken.
- **End**: Terminate the process

Below is the pseudocode for this process, followed by an explanation.

Algorithm: Pseudocode for determining if a number is even or odd.

```
1: Procedure CheckEvenOdd(number)
```

```
2: Input: number (The number to be checked)
3: Output: "Even" if number is even, "Odd" if number is odd
4: Begin
5: if (number % 2 == 0) then
6: print "Even"
7: else
8: print "Odd"
9: End if
10: End
```

Steps for Pseudocode:

1. Procedure Declaration: The pseudocode begins with the declaration of the procedure 'CheckEvenOdd' which takes a single input, 'number'.
2. Input: The procedure accepts a variable 'number' which is the integer to be checked.
3. Output: The procedure outputs "Even" if the number is even, and "Odd" if the number is odd.
4. Begin: Mark the start of the procedure.
5. Condition Check: The condition 'if (number % 2 == 0)' checks if the remainder of the number when divided by 2 is zero. The modulo operator '%' is used for this purpose.
6. Even Case: If the condition is true, the procedure prints "Even".
7. Odd Case: If the condition is false, the procedure prints "Odd".
8. End: Marks the end of the procedure

3. Define computational thinking and explain its significance in modern problem-solving. Provide examples to illustrate how computational thinking can be applied in different fields.

Ans: Definition: Computational Thinking (CT) is a problem-solving process that involves a set of skills and techniques to solve complex problems in a way that can be executed by a computer. This approach can be used in various fields such as computer science, biology, mathematics, and even daily life.

Significance:

Computational thinking involves several key principles that guide the process of problem-solving in a structured manner.

1. **Problem Understanding**: Understanding a problem involves identifying the core issue, defining the requirements, and setting the objectives. Understanding the problem is the first and most important step in problem-solving, especially in computational thinking. This involves thoroughly analyzing the problem to identify its key components and requirements before attempting to find a solution.

2. Problem Simplification: Simplifying a problem involves breaking it down into smaller, more manageable sub-problems.
Example: To design a website, break down the tasks into designing the layout, creating content, and coding the functionality.
3 Solution Selection and Design: Choosing the best solution involves evaluating different approaches and selecting the most efficient one. Designing the solution requires creating a detailed plan or algorithm.

Examples of Computational Thinking in Different Fields

1. Healthcare

Application: Disease Diagnosis
Using CT, medical data can be analyzed to identify patterns in symptoms and predict diseases. For instance, CT is applied in designing AI algorithms for imaging analysis in radiology, detecting anomalies like tumors.
Significance: Enhances accuracy and efficiency in medical diagnoses.

2. Education

Application: Personalized Learning Systems
By analyzing student performance data, CT helps design adaptive learning platforms that tailor educational content to individual needs.
Significance: Improves learning outcomes through data-driven insights.

3. Business

Application: Fraud Detection
CT aids in developing algorithms to recognize unusual transaction patterns, flagging potential fraud in financial systems.
Significance: Reduces financial losses and enhances security.

4. Discuss the concept of decomposition in computational thinking. Why is it important?

Ans: Decomposition is the method of breaking down a complicated problem into smaller, more convenient components. Decomposition is an important step in computational thinking. It involves dividing a complex problem into smaller, manageable tasks. Let's take the example of building a birdhouse. This task might look tough at first, but if we break it down, we can handle each part one at a time. Here's how we can decompose the task of building a birdhouse.

- **Design the Birdhouse**: Decide on the size, shape, and design. Sketch a plan and gather all necessary measurements.
- **Gather Materials**: List all the materials needed such as wood, nails, paint, and tools like a hammer and saw.
- **Cut the Wood:** Measure and cut the wood into the required pieces according to the design. • **Assemble the Pieces:** Follow the plan to assemble the pieces of wood together to form the structure of the birdhouse.

• **Paint and Decorate:** Paint the birdhouse and add any decorations to make it attractive for birds.

• **Install the Birdhouse:** Find a suitable location and securely install the birdhouse where birds can easily access it.

5. Explain pattern recognition in the context of computational thinking. How does identifying patterns help in problem-solving?

Ans: Pattern recognition involves looking for similarities or patterns among and within problems. For instance, if you notice that you always forget your homework on Mondays, you might recognize a pattern and set a reminder specifically for Sundays. Pattern recognition is an essential aspect of computational thinking. It involves identifying and understanding regularities or patterns within a set of data or problems. Let's consider the example of recognizing patterns in the areas of squares. The upper row represents the side lengths of squares, ranging from 1 to 7. The lower row shows the corresponding areas of these squares. Here, we can observe a pattern in how the areas increase.

• Side Length 1: Area = $1^2 = 1$
• Side Length 2: Area = $2^2 = 4$ (1 + 3)
• Side Length 3: Area = $3^2 = 9$ (1 + 3 + 5)
• Side Length 4: Area = $4^2 = 16$ (1 + 3 + 5 + 7)
• Side Length 5: Area = $5^2 = 25$ (1 + 3 + 5 + 7 + 9)
• Side Length 6: Area = $6^2 = 36$ (1 + 3 + 5 + 7 + 9 + 11)
• Side Length 7: Area = $7^2 = 49$ (1 + 3 + 5 + 7 + 9 + 11 + 13)

We can see that the area of each square can be calculated by adding consecutive odd numbers. For example, the area of a square with a side length of 3 can be found by adding the first three odd numbers: 1 + 3 + 5 = 9.

6. What is an abstraction in computational thinking? Discuss its importance and provide examples of how abstraction can be used to simplify complex problems.

Ans: Abstraction:

• **Definition:** Abstraction is the process of hiding the complex details while exposing only the necessary parts.

• **Importance:** Abstraction is a fundamental concept in problem solving, especially in computer science. It involves simplifying complex problems by breaking them down into smaller, more manageable parts, and focusing only on the essential details while ignoring the unnecessary ones. This helps in understanding, designing, and solving problems more efficiently. It helps reduce complexity by allowing us to focus on the high-level overview without getting lost in the details.

• **Example:**

Making a Cup of Tea - High-level Steps:

1. Boil water.

2. Add tea leaves or a tea bag.
3. Steep for a few minutes.
4. Pour into a cup and add milk/sugar if desired.

7. Describe what an algorithm is and explain its role in computational thinking. Provide a detailed example of an algorithm for solving a specific problem, and draw the corresponding flowchart.

Ans: An algorithm is a step-by-step collection of instructions to solve a problem or complete a task similar to following a recipe to bake a cake.. An algorithm is a precise sequence of instructions that can be followed to achieve a specific goal, like a recipe or a set of directions that tells you exactly what to do and in what order.

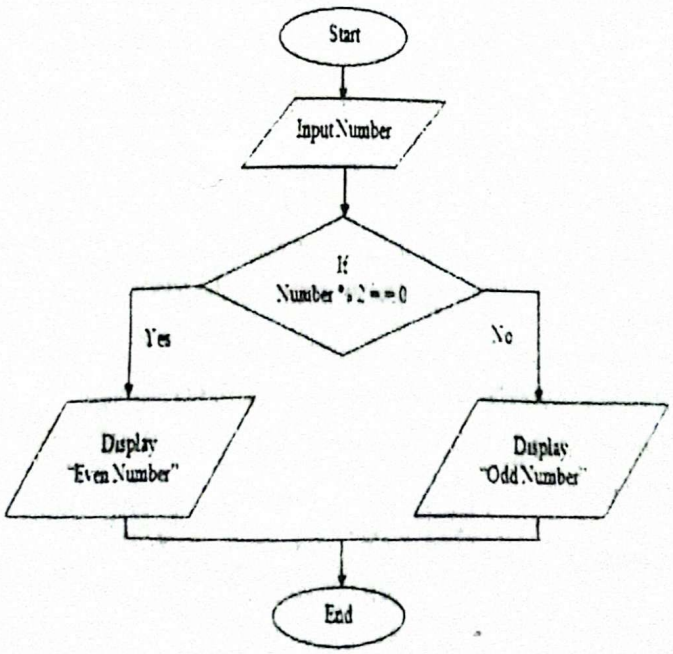
Role of Algorithms in Computational Thinking

- Using algorithm, a complex problem is broken into smaller and manageable steps.
- It enables machines to perform tasks without human involvement.
- It creates clear instructions for solving problems.
- It helps in finding efficient solutions.

Algorithm to find whether a number is even or odd:

Step 1: Start
Step 2: Read a number to N
Step 3: Divide the number by 2 and store the remainder in R.
Step 4: If R = O Then go to Step 6
Step 5: Print "N is odd" go to step 7
Step 6: Print "N is even"
Step 7: Stop

Flowchart:



8. Compare and contrast flowcharts and pseudocode as methods for algorithm design. Discuss the advantages and disadvantages of each method, and provide examples where one might be preferred over the other.

Ans:

Flowcharts are visual representations of the steps in a process using different symbols connected by arrows. They are widely used in various fields, including computer science, engineering, and business, to design systems, and communicate complex workflows clearly and effectively. **Pseudocode** is a method of representing an algorithm using simple and informal language that is easy to understand. It combines the structure of programming clarity with the readability of plain English, making it a useful tool for planning and explaining algorithms.

Pseudocode	Flowcharts
· Pseudocode uses plain language and structured format to describe the steps of an algorithm. · It is read like a story, with each step is written out sequentially. · Pseudocode communicates the steps in a detailed, narrative -like format. · It is particularly useful for documenting algorithms and can be converted into actual code in any programming language.	· Flowcharts use graphical symbols and arrows to represent the flow of an algorithm. · It is like watching a movie, where each symbol (such as rectangles, diamonds, and ovals) represents a different type of action or decision, and arrows indicate the direction of the flow. · Flowchart communicates the process in a visual format by which we can understand the overall flow easily. · They are useful for identifying the steps and decisions in an algorithm at a glance.

Flowcharts

Advantages:

- **Easy to draw.**
- **Easy to understand problem solving.**
- **Easy to identify errors.**
- **Easy to observe flow from one step to the other.**

Disadvantages:

- More time is required to draw a flowchart.
- Modifying a flowchart is not very easy every time.

- It is not suitable for very large problems.

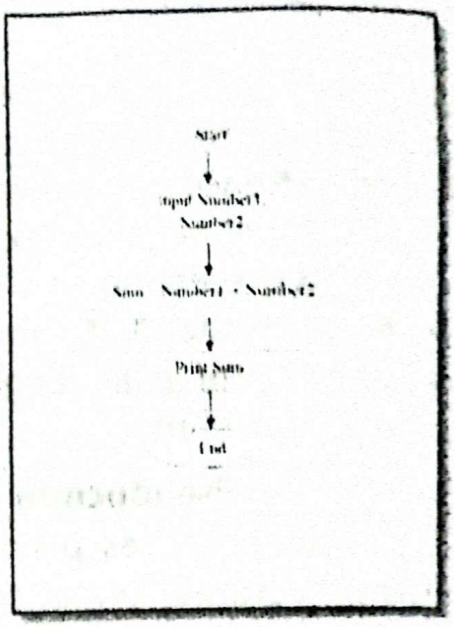
Pseudocode

Advantages:

- Easy to translate into actual programming code.
- Can describe complex algorithms compactly.
- Quick to write and modify, making it ideal for iterative design.

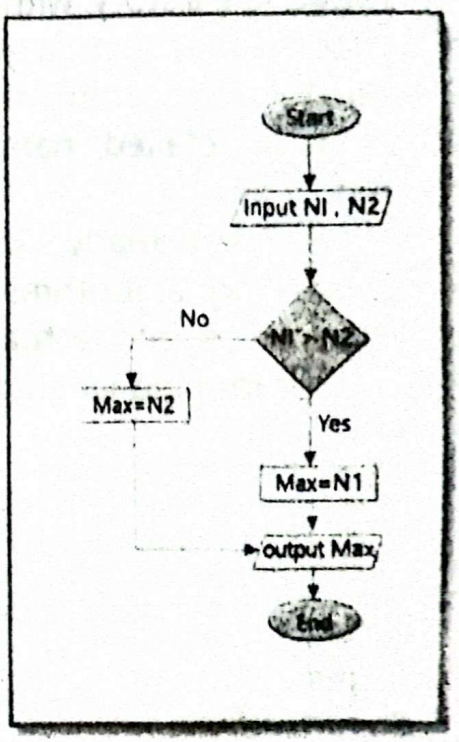
Disadvantages:

- Not as intuitive for non-technical audiences.
- If not standardized, can be interpreted differently by readers.
- Requires familiarity with logical constructs and syntax.



The "better" choice :

Flowcharts are better for non-technical or visual learners, while pseudocode is better for developers and technical teams. In practice, both tools can complement each other. For instance, you might use a flowchart to explain the overall logic and pseudocode to define detailed steps.



9. Explain the concept of a dry run in the context of both flowcharts and pseudocode. How does performing a dry run help in validating the correctness of an algorithm?

Ans: <u>Dry Run of a Flowchart:</u>

A dry run of a flowchart involves manually go through the flowchart step-by-step to understand how the algorithm works without using a computer. This helps to identify any logical errors and understand the flow of control.

Example: Calculating the Sum of Two Numbers

Consider the flowchart for adding two numbers:

Steps to dry run this flowchart:

1. Start
2. Input the first number (e.g., 3)
3. Input the second number (e.g., 5)
4. Add the two numbers (3 + 5 = 8)
5. Output the result (8)
6. Stop

Dry Run of Pseudocode

A dry run of pseudocode involves manually simulating the execution of the pseudocode line-by-line. This helps in verifying the logic and correctness of the algorithm.

Example: Finding the Maximum of Two Numbers

Consider the pseudocode for finding the maximum of two numbers:

Algorithm: FindMax

1. Input: num1, num2
2. if num1 > num2 then
3. max = num1
4. else
5. max = num2
6. end if
7. Output: max

Steps to dry run this pseudocode:

1. Input num1 and num2 (e.g., 10 and 15)
2. Check if num1 > num2 (10>15: False)
3. Since the condition is False, max = num2 (max = 15)
4. Output max (15)

Conclusion:

Performing a dry run helps in validating the correctness of an algorithm in following ways:

•Dry running your code or algorithm helps catching errors early in the development process, saving time and effort.

• Many professional programmers and computer scientists use dry running as a debugging technique to ensure their algorithms work correctly

•A dry run helps validate the correctness of algorithms designed as flowcharts or pseudocode by ensuring the logic and execution paths produce the expected results. It is a critical step in debugging and refining algorithms before implementation.

10. What is LARP? Discuss its importance in learning and practicing algorithms.

Ans: LARP stands for Logic of Algorithms for resolution of Problems. It is a fun and interactive way to learn how algorithms work by actually running them and

seeing the results. Think of it as a playground where you can experiment with different algorithms and understand how they process data.

Importance of LARP in Learning and Practicing Algorithms

LARP helps you:

- Understand how algorithms work.
- Observe the effect of different inputs on the output.
- Practice writing and improving your own algorithms.
- Clear representations (like pseudocode and flowcharts) make it easier to identify and fix errors.
- Encourages a systematic approach, reducing errors.
- Breaks down the process into manageable steps.

11. How does LARP enhance the understanding and application of computational thinking principles? Provide a scenario where LARP (Logic of Algorithms for resolution of Problems) can be used to improve an algorithm.

Ans: LARP can be used to enhance the understanding and application of computational thinking principles by providing an interactive learning experience.

Computational Thinking Principles

1. Decomposition: Breaking down complex problems into smaller, manageable parts.

2. Pattern Recognition: Identifying patterns and relationships within data.

3. Abstraction: Focusing on essential features while ignoring non-essential details.

4. Algorithmic Thinking: Developing step-by-step instructions to solve a problem.

Suppose we have a simple program that checks whether a given number is even or odd:

Initial Algorithm

- Use the modulus operator (%) to determine if the number is divisible by 2:
- If num % 2 == 0,
- The number is even.
- Otherwise, it is odd.

Pseudocode:

```
Input: num
If (num % 2) == 0:
Output "Even"
Else:
Output "Odd"
```

Limitations

1. Inefficient use of modulus operator: The modulus operator (%) can be expensive in terms of computation.

2. Lack of input validation: The program does not check if the input is a valid integer.

Applying LARP

To improve the algorithm, we can apply LARP:

1. Decomposition: Break down the problem into smaller sub-problems:
   - Checking if a number is even or odd
   - Validating user input
2. Pattern Recognition: Identify patterns in the data:
   - Even numbers always have a 0, 2, 4, 6, or 8 in the ones place
3. Abstraction: Focus on essential features:
   - Checking the last digit of the number
4. Algorithmic Thinking: Develop a new algorithm that includes the perceptions gained from decomposition, pattern recognition, and abstraction:
   - Use bitwise operations to check if the last digit is even or odd

Improved Algorithm:

- Input the number.
- Use the bitwise AND operation to check the LSB.
- Output "Even" if the result is 0 and "Odd" otherwise.

Pseudocode:

- Input: num
- If (num & 1) == 0:
- Output "Even"
- Else:
- Output "Odd"

NOTE:

Time Complexity for both the modulus and bitwise approaches, but the bitwise method is more efficient in terms of processing speed.

Additional Multiple Choice Questions (MCQs)

1. What is computational thinking primarily used for?

a) Writing novels b) Solving complex problems
c) Drawing diagrams d) Playing video games

2. Which of the following is NOT a component of computational thinking?

a) Decomposition b) Pattern recognition
c) Debugging d) Abstraction

3. **What does decomposition involve?**
a) Breaking problems into smaller tasks
b) Combining tasks into one big problem
c) Ignoring complex problems d) Analyzing algorithms

4. **In pattern recognition, which of the following is identified?**
a) Complex algorithms b) High-level abstractions
c) Similarities or patterns d) Errors in code

5. **What does abstraction focus on?**
a) Including all details b) Hiding unnecessary details
c) Combining multiple algorithms d) Recognizing errors

6. **Which of the following is an example of an algorithm?**
a) Writing a novel b) Painting a picture
c) Recipe of Baking a cake d) Observing patterns in data

7. **What is the first step in planting a tree algorithm?**
a) Digging a hole b) Choosing a suitable spot
c) Watering the tree d) Adding mulch

8. **What is the first step in computational thinking?**
a) Problem simplification b) Solution design
c) Understanding problem d) Writing algorithms

9. **Which of the following is NOT a benefit of problem understanding?**
a) Gaining clarity and focus b) Defining goals
c) Avoiding mistakes d) Skipping problem analysis

10. **In the example of building a school website, which aspect is NOT part of understanding the problem?**
a) Identifying user needs b) Coding immediately
c) Determining technical constraints d) Listing the required features

11. **What is the purpose of solution design?**
a) To test multiple solutions b) To create a detailed plan or algorithm
c) To skip understanding the problem d) To directly implement the solution

12. **What is a key factor in building an effective school website?**
a) Coding immediately b) Ignoring user needs
c) Understanding requirements d) Only focusing on visuals

13. **Which symbol in a flowchart represents a decision point?**
a) Oval b) Rectangle
c) Parallelogram d) Diamond

14. **What does a parallelogram symbolize in a flowchart?**
a) Start of a process b) End of a process
c) Input/output d) A decision point

15. What is NOT an advantage of using pseudocode?
a) It helps plan the algorithm.
b) It provides a universal way to convey steps.
c) It is directly executable by a computer.
d) It ensures algorithm logic is sound.

16. Who popularized flowcharts in the early days of computing?
a) Alan Turing b) John von Neumann and Herman Goldstine
c) Grace Hopper d) Charles Babbage

17. In a flowchart, which symbol is used to represent a process?
a) Oval b) Rectangle c) Diamond d) Arrow

18. What is the condition to check if a number is even in pseudocode?
a) if (number % 3 == 0) b) if (number / 2 == 0)
c) if (number % 2 == 0) d) if (number * 2 == 0)

19. What does time complexity measure in an algorithm?
a) Space usage b) Running time
c) Input size d) Output size

20. Which notation is commonly used to express time complexity?
a) Big Ω b) Big Θ c) Big O d) Big Δ

21. What does time complexity measure?
a) Amount of memory b) Correctness of an algorithm
c) Change in running time with input size
d) The number of errors in an algorithm

22. Which notation is used to express time complexity?
a) P-notation b) Big-O notation
c) Alpha notation d) Sigma notation

23. What does $O(n^2)$ time complexity indicate?
a) Constant time b) Linear time
c) Quadratic time d) Logarithmic time

24. If an algorithm's time complexity is O(logn), what type of growth does it have?
a) Exponential b) Logarithmic c) Linear d) Quadratic

25. Why is space complexity important in algorithm design?
a) To minimize the running time
b) To optimize the use of memory resources
c) To increase algorithm complexity
d) To improve input accuracy

26. Which of the following represents the fastest time complexity?
A) $O(n^2)$ b) O(logn) c) O(n) d) O(1)

27. What is the primary focus when evaluating an algorithm?
a) The programming language used b) The syntax of the algorithm
c) The time and space complexity d) The number of inputs

28. Which of the following represents space complexity?
a) Time taken to process an input b) Total memory used by an algorithm
c) Number of operations in an algorithm d) Number of input variables

29. What is the purpose of a dry run?
a) To execute the algorithm
b) To manually test an algorithm sample data
c) To create pseudocode for the algorithm
d) To optimize the code for performance

30. Which of the following is NOT a benefit of simulation?
a) Cost-effective b) Unsafe for real-world application
c) Repeatable for multiple tests d) Allows testing of dangerous situations

31. What is an example of simulation?
a) Writing pseudocode for an algorithm
b) Walking through a flowchart manually
c) Using a computer model for traffic flow analysis
d) Debugging errors in code

32. In the flowchart example for adding two numbers, what is the last step?
a) Input the first number b) Display the sum
c) Input the second number d) Stop

33. What is LARP used for?
a) To design algorithms b) To test algorithms
c) To learn how algorithms work d) To debug algorithms

34. Which of the following is a type of error in algorithms?
a) Syntax Error b) Runtime Error
c) Logical Error d) All of the above

35. What is the purpose of debugging?
a) To design algorithms b) To test algorithms
c) To find and fix errors in algorithms d) To learn how algorithms work

36. What does the LARP methodology focus on?
a) Writing complex programs b) Visualizing algorithms and their steps
c) Debugging code errors d) Optimizing time complexity

37. What symbol is used in a flowchart to represent decision-making?
a) Rectangle b) Diamond c) Circle d) Parallelogram

38. What command is used in LARP to display messages?
a) PRINT b) WRITE c) OUTPUT d) DISPLAY

39. Which of the following is NOT a type of error in algorithms?
a) Syntax Error b) Logical Error c) Debugging Error d) Runtime Error

40. What is a common debugging technique?
a) Writing complex algorithms b) Ignoring error messages
c) Tracing steps in the algorithm d) Removing conditions from the code

41. What type of error occurs when a variable is used without being defined?

a) Syntax Error
b) Logical Error
c) Runtime Error
d) Missing Step Error

42. What is the role of IF...THEN...ELSE in LARP?

a) To handle input values
b) To provide decision-making
c) To end the algorithm
d) To repeat steps in the algorithm

43. Which debugging technique involves breaking down a problem into smaller parts?

a) Simplify the Problem
b) Check Conditions
c) Use Comments
d) Trace the Steps

44. What happens if a division by zero is attempted in an algorithm?

a) Logical Error
b) Runtime Error
c) Syntax Error
d) Missing Step Error

Answers:

1	2	3	4	5	6	7	8	9	10	11	12
B	C	A	C	B	C	B	C	D	B	B	C
13	14	15	16	17	18	19	20	21	22	23	24
D	C	C	B	B	C	B	C	C	B	C	B
25	26	27	28	29	30	31	32	33	34	35	36
B	D	C	B	B	B	C	D	C	D	C	B
37	38	39	40	41	42	43	44				
B	B	C	C	C	B	A	B				

Topic Wise Additional Short Questions and Answers

7.1- Definition of Computational Thinking

1. What is computational thinking?

Ans: Computational Thinking (CT) is a problem-solving process that involves a set of skills and techniques to solve complex problems in a way that a can be executed by a computer. This approach can be used in various fields beyond computer science, such as biology, mathematics, and even daily life

2. What are the four key components of computational thinking?

Ans: The four key components are decomposition, pattern recognition, abstraction, and algorithms.

3. What is decomposition in computational thinking?

Ans: Decomposition is the method of breaking down a complicated problem into smaller, more convenient components. Decomposition is an important

step in computational thinking. It involves dividing a complex problem into smaller, manageable tasks

4. What is pattern recognition in computational thinking?

Ans: Pattern recognition involves looking for similarities or patterns among and within problems. Pattern recognition is an essential aspect of computational thinking. It involves identifying and understanding regularities or patterns within a set of data or problems.

5. What is abstraction in computational thinking?

Ans: Abstraction is the process of hiding the complex details while exposing only the necessary parts. It helps reduce complexity by allowing us to focus on the high-level overview without getting lost in the details.

6. What is an algorithm in computational thinking?

Ans: An algorithm is a step-by-step collection of instructions to solve a problem or complete a task. An algorithm is a precise sequence of instructions that can be followed to achieve a specific goal.

7. Give an example of an algorithm used in daily life.

Ans: Following a recipe to bake a cake is an example of an algorithm. The recipe provides a list of ingredients and step-by-step instructions to mix them and bake the cake.

7.2- Principles of Computational Thinking

8. What is the first step in computational thinking?

Answer: Problem understanding.

9. What does Problem understand mean?

Ans: Understanding a problem involves identifying the core issue, defining the requirements, and setting the objectives. This involves thoroughly analyzing the problem to identify its key components and requirements before attempting to find a solution.

10. Why is problem understanding important?

Answer: It provides clarity and focus, defines goals, leads to efficient solutions and avoids mistakes.

11. What is the role of defining goal in problem understanding?

Ans: Proper understanding of the problem allows you to define clear and achievable goals. You can determine what the final outcome should look like and set specific objectives to reach that outcome.

12. How does problem understanding help to find efficient solution?

Ans: When you comprehend the problem well, you can devise more efficient and effective solutions. You can choose the best methods and tools to address the problem, saving time and resources.

13. Which foundational step will lead to better and more effective solutions?

Ans: Always take time to thoroughly understand a problem before starting to solve it. Ask questions, gather information, and clarify any doubts. This foundational step will lead to better and more effective solutions.

14. Which steps should keep in mind before writing code of a problem?

Ans: 1. Identify Requirements 2. User Needs 3. Technical Constraints

15. What are the key principles that guide the process of problem-solving in a structured manner:

Ans: Computational thinking involves several key principles that guide the process of problem-solving in a structured manner.

• Problem understanding • Problem simplification • Solution selection and design

16. What is problem simplification in computational thinking?

Ans: Simplifying a problem involves breaking it down into smaller, more manageable sub-problems. Example: To design a website, break down the tasks into designing the layout, creating content, and coding the functionality.

17. What is solution selection and design in computational thinking?

Ans: Choosing the best solution involves evaluating different approaches and selecting the most efficient one. Designing the solution requires creating a detailed plan or algorithm.

18. What is solution selection?

Ans: Evaluating different approaches to choose the most efficient and effective solution.

19. What is involved in solution design?

Ans: Creating a detailed plan or algorithm for implementing the chosen solution.

20. What did Albert Einstein emphasize about problem understanding?

Ans: He suggested spending most of the time understanding the problem and less time thinking about solutions.

7.3- Algorithm Design Methods

21. What are algorithm design methods?

Ans: Algorithm design methods provide a range of tools and techniques to tackle various computational problems effectively. Each method has its strengths and weaknesses, making it suitable for different types of problems.

22. What are flowcharts?

Ans: Flowcharts are visual representations of the steps in a process or system, depicted using different symbols connected by arrows.

23. What is the purpose of flowcharts?

Ans: They are widely used in various fields, including computer science, engineering, and business, to model processes, design systems, and communicate complex workflows clearly and effectively.

24. What are the benefits of using flowchart?
Ans: Clarity, communication, problem solving and documentation.
25. When were the first standardized flowchart symbols developed? OR Who developed the first standardized flowchart symbols?
Ans: The first standardized flowchart symbols were developed in 1947 by the American National Standards Institute (ANSI).
26. Name any two standard symbols used in flowcharts and their purposes.
Ans:
- **Oval (Terminal):** Represents the start or end of a process.
- **Rectangle (Process):** Represents a task or operation.
- **Parallelogram (Input/Output):** Represents data input or output.
- **Diamond (Decision):** Represents a decision point with branching.
- **Arrow (Flowline):** Indicates the direction of flow.

27. Which computer scientists popularize the flowcharts?
Ans: Flowcharts were popularized by computer scientists such as John von Neumann and Herman Goldstine in the early days of computing.
28. What is Pseudocode?
Ans: Pseudocode is not actual code that can be run on a computer, but rather a way to describe the steps of an algorithm in a manner that is easy to follow.
29. What are the benefits of using pseudocode?
Ans: It helps programmers and students focus on the logic of the algorithm without worrying about the syntax of a specific programming language. It combines the structure of programming clarity with the readability of plain English, making it a useful tool for planning and explaining algorithms.
30. What is the main difference between flowcharts and pseudocode?
Ans:
- **Pseudocode:** Uses plain language to describe steps, useful for algorithm documentation.
- **Flowcharts:** Use graphical symbols for a visual representation, ideal for understanding processes at a glance.

31. Why pseudocode is used?
Ans. Using pseudocode has several benefits:
- Clarity: It helps in understanding the logic of the algorithm without worrying about syntax.
- Planning: It allows programmers to outline their thoughts and plan the steps of the algorithm.
- Communication: It is a universal way to convey the steps of an algorithm, making it easier to discuss with others.

32. Explain the purpose of a decision symbol in a flowchart.
Ans: A decision symbol represents a point where the process branches based on a yes/no or true/false condition.

33 How does pseudocode help in software development?

Ans: Pseudocode helps outline the algorithm's logic, making it easier to convert into code and communicate among team members using different programming languages.

34. What is the role of the modulo operator (%) in determining if a number is even or odd?

Ans: The modulo operator (%) checks the remainder when a number is divided by 2. If the remainder is 0, the number is even; otherwise, it's odd.

7.4- Algorithmic Activities

35. What is the purpose of design and evaluation techniques in algorithms?

Ans: To understand how efficiently algorithms solve problems.

36. What does time complexity measure in an algorithm?

Ans: Time Complexity measures how fast or slow an algorithm performs. It shows how the running time of an algorithm changes as the size of the input increases.

37. What is Big O notation used for in time complexity? OR How is time complexity commonly expressed?

Ans: Time complexity is expressed using Big-O notation, such as O(n), O(logn), or $O(n^2)$. It helps us compare different algorithms to see which one is faster.

38. Why is time complexity important?

Ans: It helps compare algorithms to determine which one is faster and more efficient for larger input sizes.

39. Why should we consider space complexity when designing algorithms?

Ans: To ensure that the algorithm can handle large inputs without exceeding memory constraints.

40. What techniques can be used to evaluate algorithms?

Ans: Techniques include dry runs and simulations to test the correctness and efficiency of algorithms.

41. Give an example of how input size affects time complexity.

Ans: Searching for a name in a list of 10 names takes less time than searching in a list of 1,000 names. Time complexity helps analyze this difference.

42. What is space complexity?

Ans: Space complexity measures the amount of memory an algorithm uses relative to input size. It is essential to consider both the memory required for the input and any extra memory used by the algorithm.

7.5- Dry Run

43. What is a dry run in the context of algorithms?
Ans: A dry run involves manually going through the algorithm with sample data to identify any errors.

44. What is the purpose of a dry run of a flowchart?
Ans: A dry run of a flowchart involves manually walking through the flowchart step-by-step to understand how the algorithm works without using a computer. This helps identify any logical errors and understand the flow of control.

45. What are the steps in performing a dry run of pseudocode?
Ans: A dry run of pseudocode involves manually simulating the execution of the pseudocode line-by-line. This helps in verifying the logic and correctness of the algorithm.

46. What is simulation in computer science?
Answer: Simulation is we use of computer programs to create a model of a real-world process or system. This helps us understand how things work by testing different ideas or algorithms without needing to try them out in real life.

47. Why is simulation used?
Ans: • ***Testing Algorithms:*** We can use simulation to see how well an algorithm works with different types of data. For example, if we want to test a new way to sort numbers, we can simulate it with different sets of numbers to see how fast it is.

• ***Exploring Scenarios:*** Simulation allows us to create many different situations to see what happens. For example, in a science experiment about plant growth, we can simulate different amounts of water or sunlight to find out which conditions help plants grow best.

48. What are some examples of simulation?
Ans: Weather forecasting, traffic flow, and scientific experiments.

49. What are benefits of simulation?
Ans: • Cost-Effective: It is often cheaper and faster to run simulations than to conduct real experiments.

• Safe: We can test dangerous situations, like a fire in a building, without putting anyone at risk.

• Repeatable: We can run the same simulation multiple times with different settings to observe how things change.

50. Give an example of a simulation application.
Ans: • ***Weather Forecasting:*** Meteorologists use simulations to predict the weather. They input data about temperature, humidity, and wind speed into a computer model to see how the weather might change over the next few days

- ***Traffic Flow:*** City planners can simulate traffic to see how changes to roads or traffic lights might affect the flow of cars. This helps them design better roads and reduce traffic jams.

51. What is the difference between a dry run and simulation?

Ans: A dry run is manual and often used for small-scale tests, while a simulation involves computer programs and is suitable for large-scale, complex scenarios.

52. What are some scenarios where simulation is useful?

Ans: Testing algorithms, weather forecasting, traffic flow analysis, and modelling dangerous situations like fires.

53. What are the steps to dry run a flowchart for adding two numbers?

Ans: 1. Start
2. Input the first number (e.g., 3)
3. Input the second number (e.g., 5)
4. Add the two numbers (3 + 5 = 8)
5. Output the result (8)
6. Stop

7.6- Introduction to LARP (Logic of Algorithms for Resolution of Problems:

54. What does LARP stand for and what is its purpose?

Ans: LARP stands for ***Logic of Algorithms for Resolution of Problems***. It is an interactive method to learn how algorithms work by running them and seeing the results.

55. Why is LARP Important?

Ans: LARP helps you:
- Understand how algorithms work.
- See the effect of different inputs on the output.
- Practice writing and improving your own algorithms.

56. What are the key components of writing algorithms in LARP?

Ans: Algorithms in LARP start with START and end with END. They use commands like WRITE for output, READ for input, and IF...THEN...ELSE for decision-making.

57. What are flowcharts in LARP used for?

Ans: Flowcharts in LARP visually represent the steps of an algorithm using standard symbols. They help learners understand the logical flow and test the algorithm step-by-step.

58. How can we draw flowcharts in LARP?

Ans: Drawing flowcharts in LARP involves steps using standard flowchart symbols such as rectangles for processes, diamonds for decisions, and parallelograms for input/output operations. Once the flowchart is created, it can be executed in LARP by translating the flowchart into LARP syntax.

59. What are the three main types of errors in algorithms?

Ans: • ***Syntax Errors:*** These occur when we write something incorrectly in our algorithm or flowchart. For example, missing a step or using the wrong symbol.

• ***Runtime Errors:*** These happen when the algorithm or flowchart is being executed. For example, trying to perform an impossible operation, such as dividing by zero.

• ***Logical Errors:*** These are mistakes in the logic of the algorithm that cause it to behave incorrectly. For example, using the wrong condition in a decision step.

60. What is meant by debugging in an algorithm?

Ans: The process of finding and fixing errors in an algorithm or flowchart.

61. What is the benefit of using comments in algorithms?

Ans: To help spot mistakes and understand what each part of the algorithm is supposed to do.

62. Mention two debugging techniques used for an algorithm.

Ans: • ***Trace the Steps:*** Go through each step of your algorithm or flowchart to see identity where it goes wrong.

• ***Use Comments:*** Write comments or notes in your algorithm to explain what each part is supposed to do. This can help you spot mistakes.

• ***Check Conditions:*** Ensure that all conditions in decision steps are correct.

• ***Simplify the Problem:*** Break down the algorithm into smaller parts and test each part separately.

63. What is the significance of error messages in debugging of an algorithm?

Ans: Error messages provide clues about where the problem is, making it easier to locate and fix errors.

64. Why are logical errors the hardest to find?

Ans: Logical errors do not stop the program from running but result in incorrect outputs, making them harder to identify.

65. What is the origin of the term "debugging"?

Ans: The term originated when a moth was found causing problems in an early computer. The moth was removed, and the process was called "debugging"

66. What is an example of a common error message in LARP?

Ans: Here are some common error messages you might see in LARP and what they mean:

• Missing Step - You probably forgot to include an important step in your algorithm.

• Undefined Variable - You are using a variable that hasn't been defined yet.

• Invalid Operation - You are trying to perform an operation that is not allowed like dividing by zero.