Unit 02
Computational Thinking and Algorithms

Multiple Choice Questions (MCQs)

MCQ	1	2	3	4	5	6	7	8	9
Answer	C	C	B	C	D	D	B	D	C

Short Response Questions (SRQs)

Q1. Differentiate between:

i. Clarity and Efficiency

Clarity	Efficiency
Clarity refers to ease to understand.	Efficiency refers to producing desired results without wasting time, space or energy.
Clarity allows us to understand the logic and steps of algorithms.	Efficiency of an algorithm depends on time taken by it & amount of memory required to run it
Clarity in an algorithm means well defined inputs, process and outputs.	Efficiency in an algorithm means producing correct results in minimum time using minimum resources.

ii. Abstraction and Pattern Recognition

Abstraction	Pattern Recognition
Abstraction refers to focus on main idea, functionality or design without getting into details	Pattern recognition refers to understanding similarities or trends in data or programs.
Abstraction is about looking into big picture without getting lost in the tiny details.	Pattern Recognition is about to reuse an existing concept rather than re-inventing it.
Abstraction reduces the complexity of viewing the things	Helps us to solve more complex problems more efficiently.

iii. Pseudocode and Flowchart

Pseudocode	Flowchart
Pseudocode is a combination of natural language and simplified programming keywords.	Flowchart is combination of symbols.
It's a more structured or semi formal way representation of algorithm	It's a pictorial representation of algorithm.
Pseudocode explains the logic and structure of Algorithm.	Flowchart explains the flow or sequence of execution.

iv. Data structure and Control Structure

Data Structure	Control Structure
Data structure is format for storing, processing and organizing data	Control Structure are statements to control sequence / flow of execution.
Data structure has many types like array, stack, queue, etc.	Control structures are of two types. Conditional statements. Loop / Repetitive statements.
Using appropriate data structure help programmer to decide how data is actually stored in computer's memory	Using appropriate Control Structure help programmer to decide which code is based on condition and which code is executed repeatedly.

v. Differentiate between Algorithm and Pseudocode.

Algorithm	Pseudocode
Algorithm is a step-by-step solution to solve a problem written in natural language.	Pseudocode is a combination of natural language and simplified programming keywords.
It's a finite set of steps that represent a solution to problem.	It's a more structured or semi formal representation of algorithm
Algorithm explains the sequence of steps involved to solve a problem.	Pseudocode explains the logic and structure of Algorithm.

Q2. Write a note on working of Bubble Sort.

Working of Bubble Sort

- To sort an array in ascending order, then the algorithm starts by taking first element of the array and compare it with the second element in first iteration.
- Suppose if the first element is greater than the second element, it will then swap these elements.
- Now move forward to compare the second element with the third element, and it continues until the largest element bubbled up to the last (Right Most) place in the array.

Q3. Write names of commonly used computational artifacts made during computational thinking.

- **Computational artifacts** refer to human made objects that helps to describe the architecture, design and function of software.
- Commonly used *computational artifacts* include meeting notes, images, prototype, workflow diagram, programs, websites and video animations.

Q4. Where we prefer to use binary search algorithm rather than linear search algorithm?

Binary Search	Linear Search
When the data in list is sorted.	When the data in list is not sorted.
Suitable for larger number of elements.	Suitable for a smaller number of elements.
More efficient and more complex.	Less efficient and less complex.

Q.5 What are the advantages of using flowcharts.

- Flowchart is a graphical representation of an algorithm therefore it is easy to understand the logic of steps to solve a particular problem.

- It helps in debugging and maintenance process of a program.
- It makes coding easy and simpler.
- It is easy for branching and looping.

Extended Response Questions (ERQs)

Q1. Determine the properties involved in computational thinking.

Computational Thinking

The characteristics that define computational thinking are

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

i. **Abstraction:** it is about looking at the big picture without getting lost in the tiny details. Abstraction is about focusing the most relevant aspects of a problem while ignoring the unnecessary details.
ii. **Decomposition**: Breaking down a complex problem into smaller, more manageable sub problems or tasks. Each part becomes a manageable sub task.
iii. **Pattern Recognition**: is about identifying patterns or trends within data, problems or solutions.
iv. **Algorithm Design**: Developing a step-by-step solution to solve a problem using a natural language. Algorithm is a finite set of steps to solve a problem.

Q2. Write an Algorithm that inputs length in inches and prints it in centimeter.

Algorithm

Step1: Start
Step2: Input length from keyboard in inches (**inches**)
Step3: Process **cm = inches * 2.54**
Step4: Output Length in centimeter (**cm**) on the screen.
Step5: Stop.

Q3. Write an algorithm to print multiplication of a number in reverse order.

Algorithm

Step1: Start
Step2: Input a number from keyboard to print its table. Let it be N
Step3: Initialize counter variable. Let K = 10
Step4: Output Number N, Counter K, Product N*K
Step5: Process K = K – 1 (Decrement K by 1)
Step6: If K > 0 then Go to Step 4 otherwise Go to step 7
Step 7: Stop.

Q.4 Explain the uses of flowcharts.

Flowcharts

Flowcharts are diagrams or pictorial representation of algorithm that describe the flow of steps to solve a particular problem.

Flowcharts Are Used For:

- **Explaining Processes:** Flowcharts are helpful to make complicated processes easy to understand by breaking them down into simple steps.

- **Finding Problems:** They help in finding errors or bugs. By looking at a flowchart, it is easy to find the part or process that is causing the error.
- **Planning Software:** In computer programming, flowcharts are used to plan out how a program will work. They show what steps need to happen and in what order.
- **Organizing Business Tasks:** Businesses use flowcharts to organize their work. For example, they can map out how orders are processed or how customer service calls are handled.
- **Learning New Things:** Flowcharts are also used in schools and training programs to teach new ideas. They make it easier to understand complex concepts by showing them visually.

Q5. A newly developed algorithm needs to be tested. Argue about the reasons.

Algorithms need to be tested for several important reasons:

1. **Accuracy**: Testing ensures that algorithms produce correct results for various inputs.
2. **Reliability**: Algorithms should work consistently.
3. **Robustness**: Algorithms should be able to handle different types of inputs.
4. **Validation**: Testing validates that the algorithm behaves as expected based on its design and specifications. It confirms that it meets the intended requirements and goals.

How to test an algorithm

Algorithms are tested by tracing through it also known as "Desk Check or Dry Run". In this process, we manually verify the algorithm whether they produce correct result or desired output. Tracing involves the following steps:

i. **Understand the Algorithm:** thoroughly read and understand the algorithm`s logic / sequence of execution (flow).
ii. **Choose Inputs:** prepare input data or scenarios for each test case. Ensure that the inputs cover a wide range of possibilities and include both valid and invalid data.
iii. **Initialization:** Initialize the variables and data structures.
iv. **Run the Algorithm:** Execute the algorithm step by step, track record inputs, update variables and follow the control structures (if-else or loops)
v. **Handle Errors:** Test the algorithm's error-handling capabilities by providing invalid inputs. Verify that the algorithm handles errors without crashing or producing incorrect results.
vi. **Repeat Until Completion:** Tracing the algorithm continues step by step until we reach the end of algorithm.
vii. **Verify Outputs:** Compare the algorithm's outputs against expected results for each test case.

Activities

Activity 1: Complete the given trace table using the following algorithm snippet.

1. Start
2. x = 2
3. total = 0
4. Repeat Step 5 Three Times
5. total = total + x
6. print(total)
7. end

Line No.	x	total	Output
2	2		
3	2	0	
4	2	0	
5	2	2	
4	2	2	
5	2	4	
4	2	4	
5	2	6	
6	2	6	6

Activity 2: Sort the array using insertion sort algorithm.

Index	0	1	2	3	4	5	6
Element	10	4	5	8	6	9	2

Initial array: [10, 4, 5, 8, 6, 9, 2]

Pass 1:

Key=4

Compare 4 with 10, move 10 to the right

Sorted array after Pass 1: [**4, 10**, 5, 8, 6, 9, 2]

Pass 2:

Sorted array after Pass 1: [**4, 10**, 5, 8, 6, 9, 2]

Key = 5

Compare 5 with 10, move 10 to right, insert 5

Sorted array after Pass 2: [4, **5, 10**, 8, 6, 9, 2]

Pass 3:

Sorted array after Pass 2: [4, **5, 10**, 8, 6, 9, 2]

Key = 8

Compare 8 with 10, move 10 to right insert 8

Sorted array after Pass 3: [4, 5, **8, 10**, 6, 9, 2]

Pass 4:

Sorted array after Pass 3: [4, 5, 8, **10, 6,** 9, 2]

Key = 6

Compare 6 with 10, move 10 to the right [4, 5, 8, **6, 10**, 9, 2]

Compare 6 with 8, move 8 to the right [4, 5, **6, 8**, 10, 9, 2]

Compare 6 with 5, no swap needed, insert 6 at location 3

Sorted array after Pass 4: [4, 5, 6, 8, 10, 9, 2]

Pass 5:

Sorted array after Pass 4: [4, 5, 6, 8, **10, 9**, 2]

Key = 9

Compare 9 with 10, move 10 to the right, insert 9

Sorted array after Pass 5: [4, 5, 6, 8, **9, 10**, 2]

Pass 6:

Sorted array after Pass 5: [4, 5, 6, 8, 9, **10, 2**]

Key = 2

Compare 2 with 10, move 10 to the right: [4, 5, 6, 8, 9, **2, 10**]

Compare 2 with 9, move 9 to the right: [4, 5, 6, 8, **2, 9**, 10]

Compare 2 with 8, move 8 to the right: [4, 5, 6, **2, 8**, 9, 10]

Compare 2 with 6, move 6 to the right: [4, 5, **2, 6**, 8, 9, 10]

Compare 2 with 5, move 5 to the right: [4, **2, 5**, 6, 8, 9, 10]

Compare 2 with 4, move 4 to the right: [**2, 4**, 5, 6, 8, 9, 10]

No more elements to compare, insert 2 at location 1

Sorted array after Pass 6: [2, 4, 5, 6, 8, 9, 10]

Final sorted array: [2, 4, 5, 6, 8, 9, 10]

Activity 3: Sort the array using Bubble sort algorithm.

Bubble Sort Algorithm

Elements	3	1	7	8	2	5	6	4	0
Index	0	1	2	3	4	5	6	7	8

Initial Array {3,1,7,8,2,5,6,4,0}

Iteration 1

i. Compare 1st element (3) with 2nd element (1), swap them. {**1,3**,7,8,2,5,6,4,0}
ii. Compare 2nd element (3) with 3rd element (7), already in correct order. {1,**3,7**,8,2,5,6,4,0}
iii. Compare 3rd element (7) with 4th element (8), already in correct order. {1,3**,7,8**,2,5,6,4,0}
iv. Compare 4th element (8) with 5th element (2), swap them. {1,3,7,**2,8**,5,6,4,0}
v. Compare 5th element (8) with 6th element (5), swap them. {1,3,7,2,**5,8**,6,4,0}

vi. Compare 6th element (8) with 7th element (6), swap them. {1,3,7,2,5,**6,8**,4,0}
vii. Compare 7th element (8) with 8th element (4), swap them. {1,3,7,2,5,6,**4,8**,0}
viii. Compare 8th element (8) with 9th element (0), swap them. {1,3,7,2,5,6,4,**0,8**}

**Array at end of 1st Iteration. **{1,3,7,2,5,6,4,0,8} **

Iteration 2

i. Compare 1st element (1) with 2nd element (3), already in correct order. {**1,3**,7,2,5,6,4,0,8}
ii. Compare 2nd element (3) with 3rd element (7), already in correct order. {1,**3,7**,2,5,6,4,0,8}
iii. Compare 3rd element (7) with 4th element (2), swap them. {1,3,**2,7**,5,6,4,0,8}
iv. Compare 4th element (7) with 5th element (5), swap them. {1,3,2,**5,7**,6,4,0,8}
v. Compare 5th element (7) with 6th element (6), swap them. {1,3,2,5,**6,7**,4,0,8}
vi. Compare 6th element (7) with 7th element (4), swap them. {1,3,2,5,6,**4**,**7**,0,8}
vii. Compare 7th element (7) with 8th element (0), swap them. {1,3,2,5,6,4,**0,7**,8}

Array at end of 2nd iteration. ** {1,3,2,5,6,**4,0,7,8} **

Iteration 3

i. Compare 1st element (1) with 2nd element (3), already in correct order. {**1,3**,2,5,6**,**4,0,7,8}
ii. Compare 2nd element (3) with 3rd element (2), swap them. {1**,2,3**,5,6**,**4,0,7,8}
iii. Compare 3rd element (3) with 4th element (5), already in correct order. {1,2,**3,5**,6**,**4,0,7,8}
iv. Compare 4th element (5) with 5th element (6), swap them. Already in correct order {1,2,3,**5,6,**4,0,7,8}
v. Compare 5th element (6) with 6th element (4), swap them. {1,2,3,5**,4,6**,0,7,8}
vi. Compare 6th element (6) with 7th element (0), swap them. {1,2,3,5**,**4,**0,6**,7,8}

Array at end of 3rd iteration. ** {1,2,3,5,**4,0,6,7,8} **

Iteration 4

i. Compare 1st element (1) with 2nd element (2), already in correct order. {**1,2**,3,5**,**4,0,6,7,8}
ii. Compare 2nd element (2) with 3rd element (3), already in correct order. {1**,2,3**,5**,**4,0,6,7,8}
iii. Compare 3rd element (3) with 4th element (5), already in correct order. {1,2,**3,5,**4,0,6,7,8}
iv. Compare 4th element (5) with 5th element (4), swap them. {1,2,3,**4,5**,0,6,7,8}
v. Compare 5th element (5) with 6th element (0), swap them. {1,2,3,4,**0,5**,6,7,8}

**Array at end of 4th iteration. ** {1,2,3,4,0,5,6,7,8} **

Iteration 5

i. Compare 1st element (1) with 2nd element (2), already in correct order. {**1,2**,3,4,0,5,6,7,8}
ii. Compare 2nd element (2) with 3rd element (3), already in correct order {1**,2,3**,4,0,5,6,7,8}
iii. Compare 3rd element (3) with 4th element (4), already in correct order {1**,**2,**3,4**,0,5,6,7,8}
iv. Compare 4th element (4) with 5th element (0), swap them. {1**,**2,3,**0,4**,5,6,7,8}

Array at end of 5th iteration. ** {1,**2,3,0,4,5,6,7,8} **

Iteration 6

i. Compare 1st element (1) with 2nd element (2), already in correct order. {**1,2**,3,0,4,5,6,7,8}

ii. Compare 2nd element (2) with 3rd element (3), already in correct order {1**,2,3**,0,4,5,6,7,8}
iii. Compare 3rd element (3) with 4th element (0), swap them. {1,2,**0,3**,4,5,6,7,8}

**Array at end of 6th iteration. ** {1,2,0,3,4,5,6,7,8} **

Iteration 7

i. Compare 1st element (1) with 2nd element (2), already in correct order. {**1,2**,0,3,4,5,6,7,8}
ii. Compare 2nd element (2) with 3rd element (0), swap them. {1,**0,2**,3,4,5,6,7,8}

**Array at end of 7th iteration. ** {1,0,2,3,4,5,6,7,8}

Iteration 8

i. Compare 1st element (1) with 2nd element (0), swap them. {**0,1**,2,3,4,5,6,7,8}

**Array at end of 8th iteration. ** Complete Sorted Array {0,1,2,3,4,5,6,7,8}

Activity 4: Binary Search Algorithm

Consider the following sorted array of seven elements and we need to find a number 9.

Elements	2	4	5	6	8	9	10
Index	0	1	2	3	4	5	6

Binary Search Algorithm

Initial Array {2, 4, 5, 6, 8, 9, 10}

Target value = 9

Perform the binary search to find 9 from the given sorted array:

Step 1:

Array: {2, 4, 5, 6, 8, 9, 10}

Middle Index= (0 + 6) / 2 = 3

Middle Element = 6

Since 9 > 6, we know that the target, if it exists, must be to the right of 6.

New Search array = {8, 9, 10}

Step 2:

Now we focus on the right half:

Array = {8, 9, 10}

Middle Index = (0 + 2) / 2 = 1

Middle Element = 9

Target 9 is equal to the middle element that is also 9.

So, the index of **9** in the sorted array {2, 4, 5, 6, 8, 9, 10} is **5**.

Activity 5: Consider the following array of eight elements and we need to find a number 9 using Linear Search Algorithm:

Elements	58	25	39	78	12	9	79	80
Index	0	1	2	3	4	5	6	7

Target value = 9

Iteration 1: Compare the first element, 58, with the target number, 9. Since 58 is not equal to 9, we move to the next element.

Iteration 2: Compare the second element, 25, with the target number, 9. Again, 25 is not equal to 9, so we proceed to the next element.

Iteration 3: Compare the third element, 39, with the target number, 9. Like before, 39 is not equal to 9, so we continue.

Iteration 4: Compare the fourth element, 78, with the target number, 9. Once more, 78 is not equal to 9, so we move on.

Iteration 5: Compare the fifth element, 12, with the target number, 9. Once again, 12 is not equal to 9, so we proceed.

Iteration 6: Compare the sixth element, 9, with the target number, 9. We found a match. Since 9 is equal to the target number, we stop the search.

Result: The target value 9 is found at **index 5** in the array.