Characteristics of algorithms- along with their corresponding pseudocode.

- The pseudocode should be structured to demonstrate algorithmic thinking without requiring specific knowledge of any programming language.

1. Start with a Brief Introduction

- Explain what pseudocode is and how it is used to represent algorithms in a way that is closer to human language than programming languages.
- Emphasize that pseudocode is not bound by any specific syntax but should still follow a logical structure.

2. Use Simple, Common Terms

Use terms, such as:

- Input and Output for data inputs and results.
- Common control structures like If-Else, For, While.
- Indent for blocks of code, just like in actual coding to show structure.

3. Choose a Familiar Problem

- Pick an example problem like sorting a list, finding the maximum number, or calculating factorials.

Characteristics of Algorithms:

1. **Input**: The algorithm should have input(s).
2. **Output**: The algorithm should produce output(s).
3. **Definiteness**: Each step must be clear and unambiguous.
4. **Finiteness**: The algorithm must terminate after a finite number of steps.
5. **Effectiveness**: Each step of the algorithm should be basic enough to be carried out in a finite amount of time.

Problem 1: **Find the Largest Number in a List**

- **Problem Statement:** Given a list of numbers, find the largest number.

```
Algorithm Find Largest
Input: A list of numbers, List[n]
Output: Largest number in the list

Step 1: Set Largest to List[1]
Step 2: For each number i from 2 to n do
        If List[i] > Largest
          Set Largest to List[i]
      End For
Step 3: Return Largest
```

Explanation:

- **Input:** List of numbers.
- **Output:** The largest number.
- **Definiteness:** Each step is well-defined.
- **Finiteness:** The loop runs only for the length of the list.
- **Effectiveness:** Each comparison and update can be done easily.

Problem 2: **Sum of First N Natural Numbers**

- **Problem Statement:** Compute the sum of the first N natural numbers.

Algorithm Sum Natural Numbers

Input: A positive integer N

Output: Sum of first N natural numbers

```
Step 1: Set Sum to 0
Step 2: For each number i from 1 to N do
            Set Sum to Sum + i
        End For
Step 3: Return Sum
```

Explanation:

Input: A positive integer N.Output: Sum of the first N natural numbers.

Definiteness: Every step is well-defined.

Finiteness: The loop runs N times.

ffectiveness: Basic addition operation is used.

Problem 3: **Check if a Number is Prime**

- **Problem Statement:** Check if a given number is prime.

```
Algorithm CheckPrime
Input: A positive integer N
Output: True if N is prime, otherwise False

Step 1: If $N <= 1$, return False
Step 2: For i from 2 to $\sqrt{N}$ do
            If $N \bmod i = 0$
                Return False
        End For
Step 3: Return True
```

Explanation:
Input: A positive integer N.
Output: A boolean value indicating whether the number is prime.
Definiteness: Each step is clear.
Finiteness: The loop runs for a finite number of times up to $\sqrt{N}$.
Effectiveness: Basic division and modulus operations are used.

Problem 4: **Reverse a String**

- **Problem Statement:** Given a string, reverse the string.

```
Algorithm Reverse String
Input: A string S of length n
Output: The reversed string

Step 1: Set ReversedString to an empty string
Step 2: For i from n down to 1 do
            Set ReversedString to ReversedString + S[i]
        End For
Step 3: Return ReversedString
```

Explanation:

Input: A string S.

Output: The reversed string.

Definiteness: Each step is well-defined.

Finiteness: The loop runs exactly n times.

Effectiveness: String concatenation is a simple operation.