Certainly! Let's break down **Question 7: Algorithm** in detail.

---

7. Algorithm in C

An **algorithm** is a **step-by-step procedure** or a set of **rules** to solve a problem. It acts as a blueprint before writing actual C code.

Why Use an Algorithm?

✓ **Systematic approach** – Helps break down a problem into simple steps.
✓ **Easy debugging** – Identifies errors before writing code.
✓ **Efficiency** – Helps optimize the program.
✓ **Better understanding** – Explains logic clearly before coding.

Characteristics of a Good Algorithm

1. **Well-defined inputs** – The algorithm should clearly specify what inputs it needs.
2. **Well-defined outputs** – The algorithm must produce a specific output.
3. **Finiteness** – It should **end after a finite number of steps**.
4. **Effectiveness** – Each step should be **simple and clear**.
5. **Language-independent** – Should be written in plain language or pseudocode.

How to Write an Algorithm?

An algorithm follows a **step-wise format**, often written in **plain English or pseudocode**.

Format of an Algorithm

```
Step 1: Start
Step 2: Input necessary values
Step 3: Process the input (Perform calculations or conditions)
Step 4: Display output
Step 5: Stop
```

Example 1: Algorithm to Find the Largest of Two Numbers

Problem Statement:

Write an algorithm to compare two numbers and print the largest.

Algorithm:

```
Step 1: Start
Step 2: Input two numbers (a, b)
```

```
Step 3: If a > b, then print "a is larger"
Step 4: Else, print "b is larger"
Step 5: Stop
```

C Code Implementation

```c
#include <stdio.h>
int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    if (a > b) {
        printf("%d is larger\n", a);
    } else {
        printf("%d is larger\n", b);
    }

    return 0;
}
```

Example 2: Algorithm to Check Even or Odd

Problem Statement:

Write an algorithm to check if a number is even or odd.

Algorithm:

```
Step 1: Start
Step 2: Input a number (num)
Step 3: If num % 2 == 0, then print "Even"
Step 4: Else, print "Odd"
Step 5: Stop
```

C Code Implementation

```c
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("Even number\n");
    } else {
        printf("Odd number\n");
    }

    return 0;
}
```

Example 3: Algorithm to Find Factorial of a Number

Problem Statement:

Write an algorithm to calculate the **factorial of a number (n!)**.

Algorithm:

```
Step 1: Start
Step 2: Input a number (n)
Step 3: Initialize fact = 1
Step 4: Repeat from i = n down to 1: fact = fact * i
Step 5: Print fact
Step 6: Stop
```

C Code Implementation

```c
#include <stdio.h>
int main() {
    int n, fact = 1;
    printf("Enter a number: ");
    scanf("%d", &n);

    for (int i = n; i >= 1; i--) {
        fact *= i;
    }

    printf("Factorial of %d is %d\n", n, fact);
    return 0;
}
```

Types of Algorithms

1. **Brute Force Algorithm** – Tries all possible solutions (e.g., Linear Search).
2. **Divide and Conquer Algorithm** – Splits the problem into smaller subproblems (e.g., Merge Sort).
3. **Greedy Algorithm** – Takes the best immediate choice (e.g., Dijkstra's Algorithm).
4. **Dynamic Programming Algorithm** – Solves subproblems and stores results to avoid repetition (e.g., Fibonacci Series).
5. **Backtracking Algorithm** – Tries all possibilities and discards wrong ones (e.g., Sudoku Solver).

Key Takeaways

✓ An **algorithm** is a **step-by-step** solution to a problem.
✓ It must have **clear inputs, outputs, and a finite number of steps**.
✓ **Writing an algorithm first** helps in better **understanding and debugging**.
✓ It can be implemented in **any programming language**, including C.

Would you like any more examples or practice problems?