Introduction to Algorithms

1 Definition of an Algorithm

An algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of specific problems or to perform a computation. In simpler terms, it is a step-by-step procedure used for calculations, data processing, and automated reasoning.

An algorithm acts as a blueprint for a program. Before writing code in a specific programming language (like C++, Python, or Java), a programmer first designs the logic of the solution in the form of an algorithm.

2 Characteristics of a Good Algorithm

For an algorithm to be considered valid and efficient, it must satisfy specific criteria. Donald Knuth, a pioneer in computer science, identified five essential properties of an algorithm:

2.1 Input

An algorithm has zero or more inputs. These are quantities that are supplied to the algorithm externally before the execution begins.

2.2 Output

An algorithm must produce at least one output. The output represents the solution to the problem and must have a specific relation to the inputs.

2.3 Definiteness

Each step of an algorithm must be clear and unambiguous. The instruction should be precise so that the reader (or the computer) understands exactly what to do. There should be no room for misinterpretation.

2.4 Finiteness

An algorithm must always terminate after a finite number of steps. A procedure that goes into an infinite loop is not an algorithm; it is merely a computational method.

2.5 Effectiveness

All operations used in the algorithm must be basic enough to be performed exactly and in a finite length of time by a person using paper and pencil. Every step must be feasible.

3 Importance of Algorithms in Problem Solving

Algorithms are fundamental to computer science because they provide a systematic way to solve problems.

- **Efficiency:** They help in finding the most efficient way to solve a problem in terms of time and memory.
- **Scalability:** A good algorithm works effectively regardless of the size of the input data.
- **Reusability:** Once an algorithm is designed, it can be applied to similar problems or implemented in various programming languages.
- **Analysis:** They allow computer scientists to predict the performance of a program before it is written.

4 Algorithm vs Program

While the terms are often used interchangeably, there is a distinct difference between an algorithm and a program.

- **Algorithm:** It is a logical design or a conceptual idea for solving a problem. It is written in natural language or pseudocode and is independent of any specific hardware or software.
- **Program:** It is the concrete implementation of an algorithm. It is written in a specific programming language and is executed by a computer.

5 Representation of Algorithms

Algorithms are generally represented in two main ways to make them understandable before coding:

5.1 Pseudocode

Pseudocode is an informal high-level description of the operating principle of a computer program. It uses the structural conventions of a normal programming language but is intended for human reading rather than machine reading. It omits details like variable declarations and system-specific syntax.

5.2 Flowcharts

A flowchart is a diagrammatic representation of an algorithm. It uses different shapes to represent different types of instructions:

- **Ovals:** Start and Stop.
- **Parallelograms:** Input and Output.
- **Rectangles:** Processing or arithmetic operations.
- **Diamonds:** Decision making (Yes/No questions).

6 Basic Structure of Algorithms

Any algorithm can be constructed using three basic control structures:

6.1 Sequence

This is the default control structure. Instructions are executed one after another in the order they appear.

6.2 Selection

Also known as branching or decision-making. The algorithm chooses which path to take based on a condition (e.g., IF-ELSE statements).

6.3 Iteration

Also known as looping. A set of instructions is repeated multiple times until a specific condition is met (e.g., FOR or WHILE loops).

7 Examples of Simple Algorithms

7.1 Algorithm to Find the Maximum of Two Numbers

1. Start.
2. Input two numbers, A and B.
3. Compare A and B.
4. If $A > B$, then print ”A is greater”.
5. Else, print ”B is greater”.
6. Stop.

7.2 Algorithm to Compute the Sum of n Numbers

1. Start.
2. Input the value of n (total numbers).
3. Initialize $Sum = 0$ and $Count = 0$.
4. Input a number, let’s call it Num.
5. Add Num to Sum ($Sum = Sum + Num$).
6. Increment $Count$ by 1 ($Count = Count + 1$).
7. Check if $Count < n$.
8. If True, go to Step 4.
9. If False, print Sum.
10. Stop.

7.3 Algorithm to Check Whether a Number is Prime

1. Start.
2. Input a number N.

3. If $N < 2$, print "Not Prime" and Stop.
4. Initialize divisor $i = 2$.
5. If $i \times i > N$, print "Prime" and Stop.
6. If N is divisible by i (remainder is 0), print "Not Prime" and Stop.
7. Increment i by 1.
8. Go to Step 5.

8 Advantages of Using Algorithms

- It provides a step-by-step solution, making the logic easy to understand.
- It acts as a blueprint, simplifying the coding process.
- It is language-independent, meaning it can be implemented in Java, C++, Python, etc.
- It helps in debugging logic errors before the code is even written.

9 Limitations of Algorithms

- Writing algorithms for very complex problems can be time-consuming and tedious.
- Understanding complex logic in flowcharts or pseudocode can sometimes be difficult.
- Branching and looping can make algorithms hard to trace manually.

10 Summary of Key Concepts

An algorithm is a finite set of clear instructions to solve a problem. A good algorithm must be definite, effective, and finite. It accepts inputs and produces outputs. Algorithms are distinct from programs; they are the logic, while programs are the implementation. They rely on sequence, selection, and iteration to control the flow of execution.

11 Practice Problems and Exercises

1. Write an algorithm to swap two numbers without using a third variable.
2. Design an algorithm to find the factorial of a given number n.
3. Create an algorithm to determine if a given year is a leap year.
4. Write an algorithm to find the largest of three distinct numbers.
5. Explain why "Finiteness" is a critical characteristic of an algorithm.