Introduction to Algorithm Design

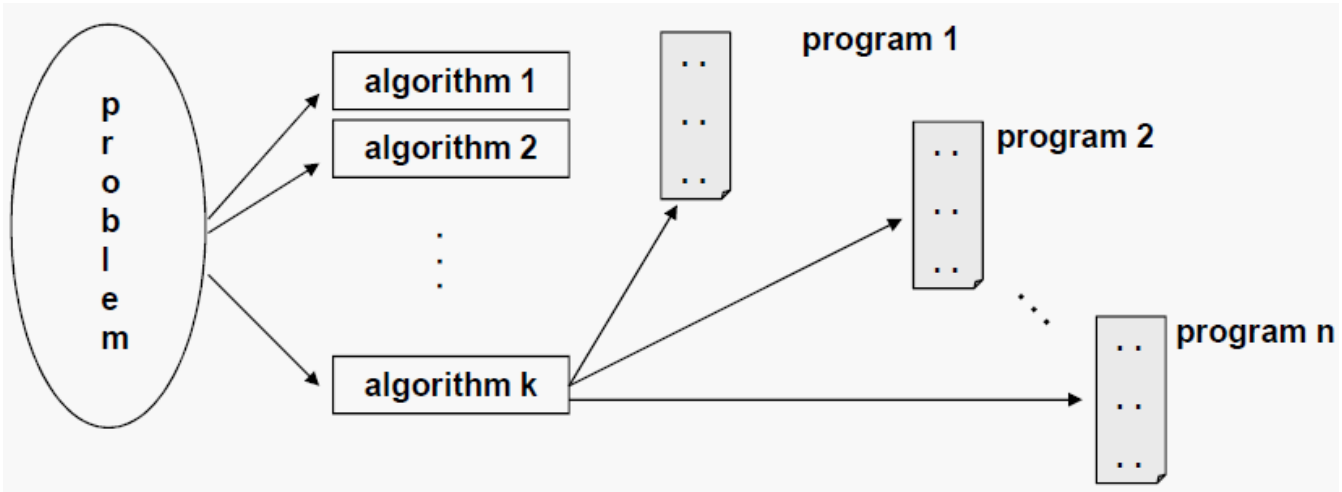
Rangaballav Pradhan

Asst. Professor, Dept. of CSE, ITER,

SOA Deemed to be University

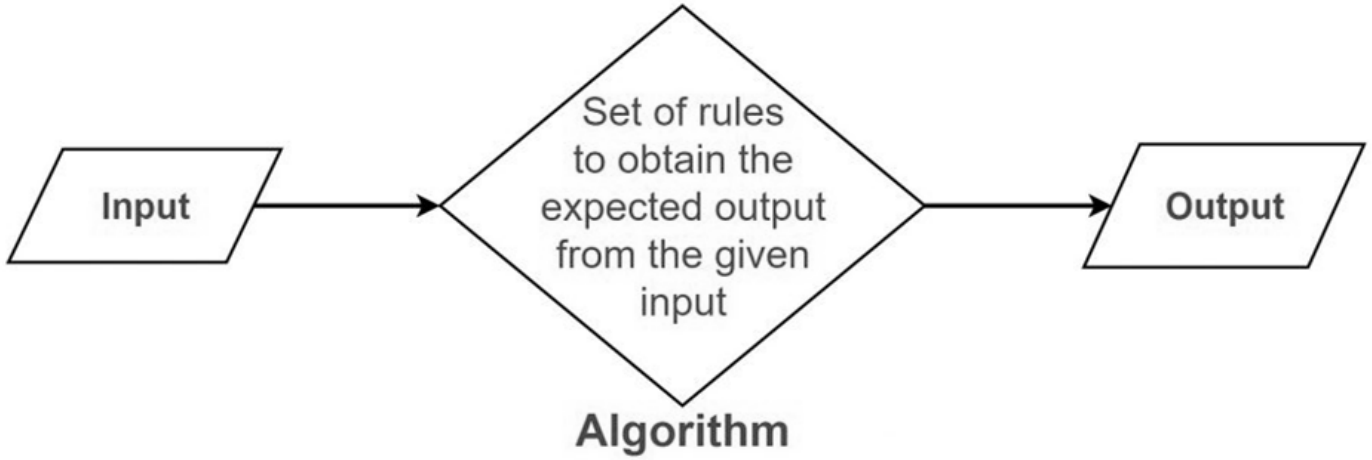
Introducing Algorithms

- What is an Algorithm?
  - Computer
  - Computing/ Data Processing
  - Complex Operations
  - Programming Language
  - A program
    - A Problem
    - An Algorithm
    - A solution



Definition

- An algorithm is a finite sequence of basic and well-defined instructions for solving a class of problems or to perform a specific computation.



Characteristics of algorithms

- Input ≥ 0
- Output ≥ 1
- Finiteness
- Definiteness
- Effectiveness
- Correctness
- Efficiency

Process of problem solving

- Define the problem (specifications, input and constraints)
- Develop a model using problem specifications and constraints
- Define the specification of an algorithm
- Design an algorithm using a suitable design approach
- Verify the correctness of the algorithm
- Analysis of the algorithm
- Implement the algorithm
- Program testing
- Documentation or publish the algorithm

Significance of algorithms

- To find the solutions for complex computational problems
- To improve the efficiency of a computer program
- Proper utilization of computing resources
- To handle the processing of larger problem instances efficiently

Example: GCD of two numbers

- The greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.
- For example, the GCD of 8 and 12 is 4, that is, gcd(8,12)=4.
- **Method 1: Using prime factorizations:**

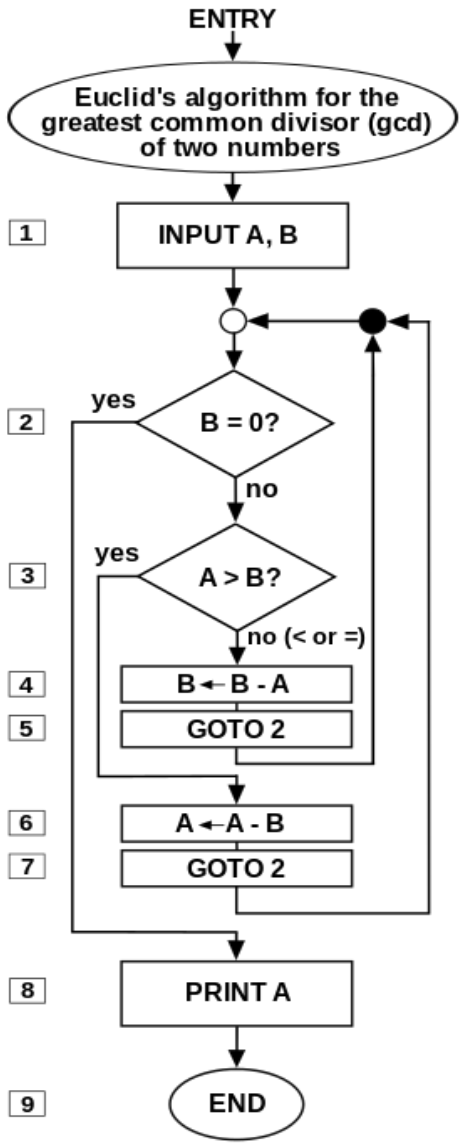
gcd (a,b)
1. Factorize a using prime factors
2. Factorize b using prime factors
3. Find the common prime factors of a and b
4. Multiply the common prime factors

gcd (48, 120)
48 = 2 x 2 x 2 x 2 x 3
120 = 2 x 2 x 2 x 3 x 5
GCD = 2 x 2 x 2 x 3
=24

Method 2: Euclidian Algorithm 1

findGCD (a, b)

- Begin
- if a = 0 OR b = 0, then
- return 0
- if a = b, then
- return b
- if a > b, then
- return findGCD(a-b, b)
- else
- return findGCD(a, b-a)
- End



- Example:

gcd (48, 120)

⇒ gcd (48, 120-48)

⇒ gcd (48, 72)

⇒ gcd (48, 72-48)

⇒ gcd (48, 24)

⇒ gcd (24, 24)

⇒ gcd (24, 0)

⇒ return 24

⇒ gcd = 24

Method 3: Euclidian Algorithm 2

GCD (a, b)

1. If a = 0 then
2. GCD(a, b) = b
3. If b = 0 then
4. GCD(a, b) = a
5. GCD(a, b) = GCD(b, a % b)

- Example:

  gcd (48, 120)

  ⇒ gcd (120, 48)

  ⇒ gcd (48, 24)

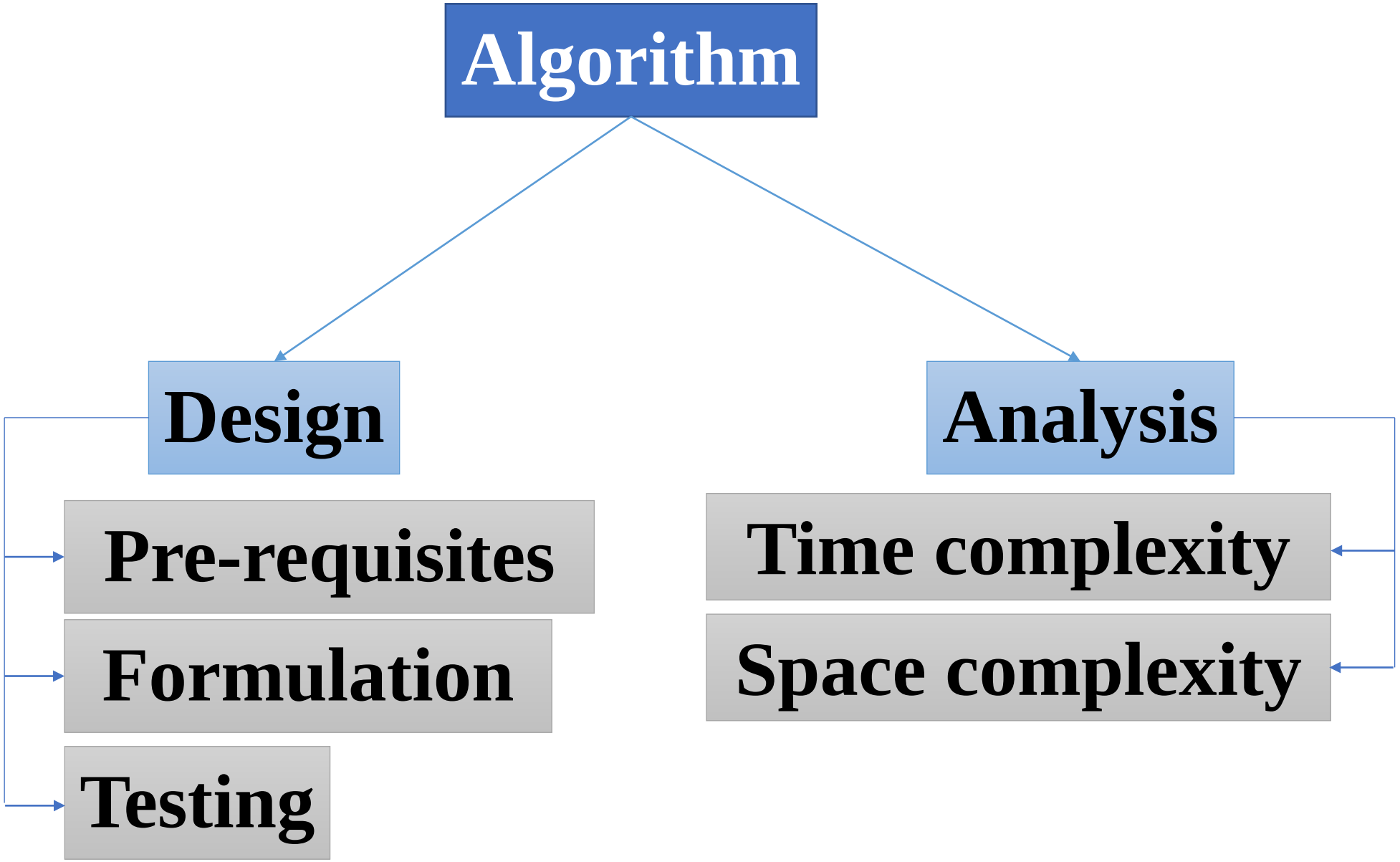
  ⇒ gcd (24, 0)

  ⇒ gcd = 24

Algorithm Writing Formats

- **English-like:** Using some natural language to explain the steps
- **Flowchart:** Using diagrammatic representation of the steps
- **Pseudocode:** Using natural language, symbolic abbreviations and mathematical notations to write a programming language like structure but not exactly a program.

```
for i ← 1 to length(A)
   x ← A[i]
   j ← i
   while j > 0 and A[j-1] > x
      A[j] ← A[j-1]
      j ← j - 1
   A[j] ← x
```


Algorithm
Design
Analysis
Pre-requisites
Formulation
Testing
Time complexity
Space complexity

Algorithm Design: Pre-requisites

- The problem definition that is to be solved by the algorithm.
- The constraints of the problem that must be considered while solving the problem.
- The input to be taken to solve the problem instance.
- The output to be expected when the problem is solved.
- The design approach to be followed.

Algorithm Design: Formulation/Definition

- Various design approaches exist:
    1. Brute force
    2. Divide and conquer
    3. Greedy algorithm
    4. Dynamic programming
    5. Backtracking algorithm
    6. Branch and Bound

Algorithm Design: Testing/Implementation

- **Recursive vs Iterative:** A recursive algorithm calls itself again and again until a base condition is met whereas iterative algorithms use loops to solve any problem.
- **Exact vs Approximate:** Algorithms that are capable of finding an exact optimal solution for a problem are known as the exact algorithm. For some problems, it is not possible to find the most optimized solution. In such cases an approximation algorithm is used that finds an approximate solution to a problem which may not be optimal but is near optimal.
- **Serial vs Parallel vs Distributed:** In serial algorithms, one instruction is executed at a time while parallel algorithms divide the problem into subproblems and execute them on different processors. If parallel algorithms are distributed on different machines over a network, then they are known as distributed algorithms.

Analysis of Algorithm

- Analysis refers to the computation of complexity of a given algorithm
- Analysis gives a measure of efficiency of the algorithm in terms of resource consumption
- Mainly two parameters are considered for analysis of algorithms: the amount of running time and the memory space.
- The running time of an algorithm is expressed as a function of the input size and is known as time complexity and the volume of memory space consumed is known as space complexity.

Analysis of Algorithm

- Priori Analysis: A priori analysis means checking the algorithm before its implementation. In this, the algorithm is checked when it is written in the form of theoretical steps. The efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation. It is in this method, that the Algorithm Complexity is determined.
- Posterior Analysis: A posterior analysis means checking the algorithm after its implementation. In this, the algorithm is checked by implementing it in any programming language and executing it. This analysis helps to get the actual and real analysis report about correctness, space required, time consumed etc.