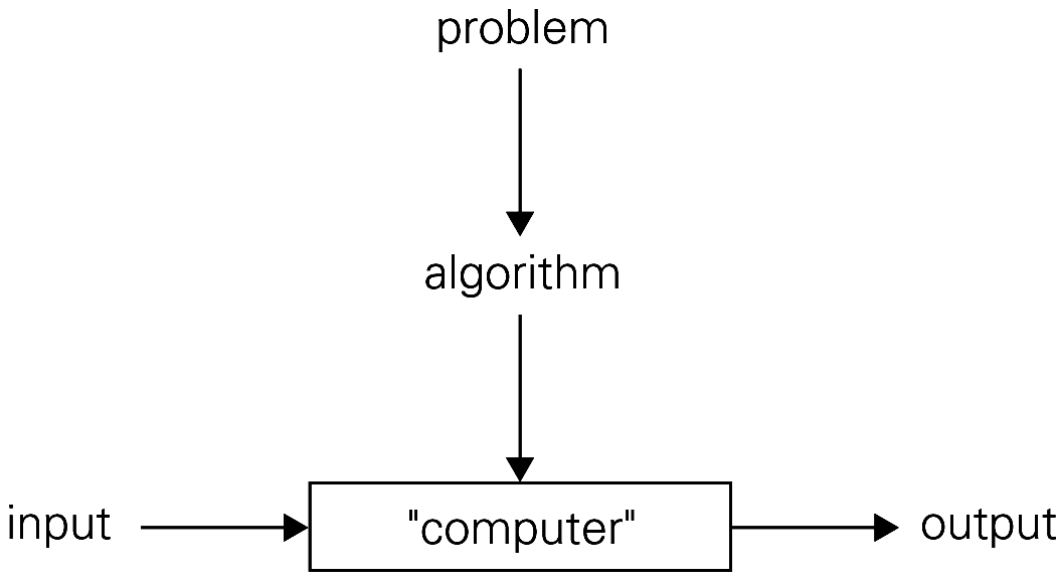
Introduction

What is an algorithm?

- A sequence of unambiguous instructions for solving a problem



Why study algorithms?

- As a computer professional:
  - Need to know a standard set of important algorithms
  - Be able to design and evaluate new algorithms
  - The study of algorithms is the core of computer science
- It is indispensable in almost all aspects of our lives
- It is useful for us to developing analytical skills

Important Points to Remember

- Nonambiguity requirement
- Carefully specify the range of inputs
- Same algorithm can have different representations
- Same problem can be solved with different algorithms that may have dramatically different speeds

An example

- **Problem**:
  computing the greatest common divisor of two integers, m and n.

- **Consecutive integer checking algorithms**:
  based on the definition – the largest integer that divides m and n evenly

Consecutive integer checking algorithms

Step 1	Assign the value of min(m,n) to t
Step 2	Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4
Step 3	Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4
Step 4	Decrease t by 1 and go to Step 2

An example

- **Problem**:
  computing the greatest common divisor of two integers, m and n.

- **Middle-school procedure:**
  Using prime factorization

Middle-school procedure

Step 1	Find the prime factorization of m
Step 2	Find the prime factorization of n
Step 3	Find all the common prime factors
Step 4	Compute the product of all the common prime factors and return it as gcd(m,n)

An example

- **Problem**:
  computing the greatest common divisor of two integers, m and n.

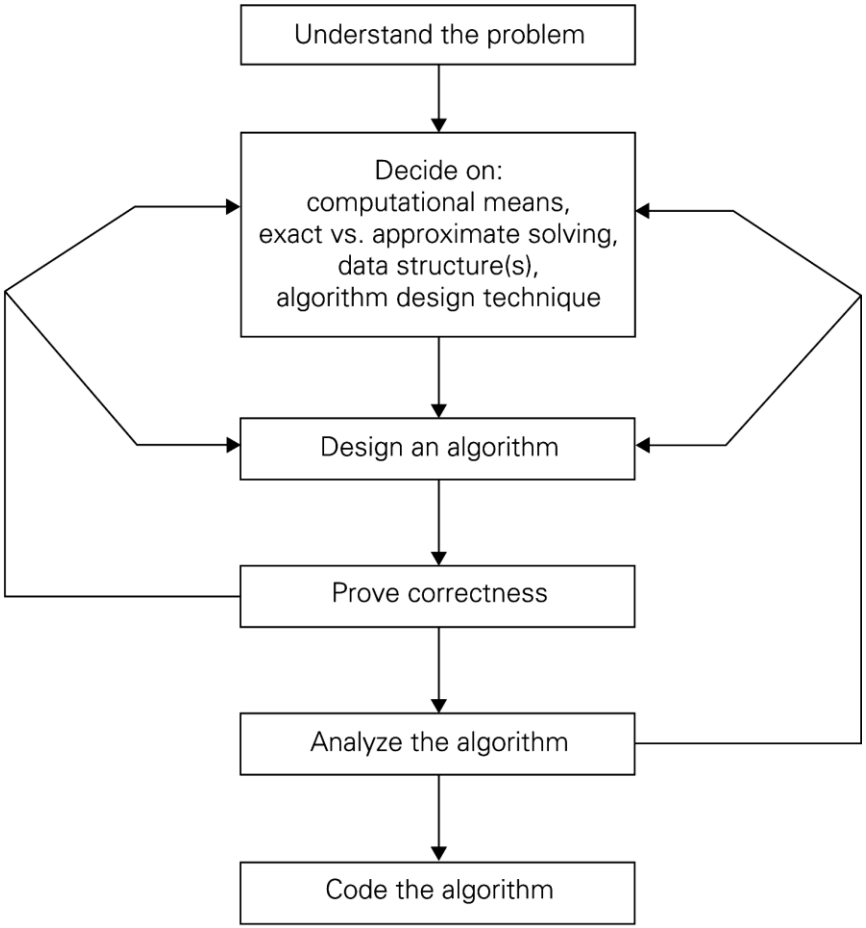
- **Euclid's algorithm**:
  $\gcd(m, n) = \gcd(n, m \bmod n)$
  $\gcd(m, 0) = m$

Euclid's algorithm

Step 1: If n = 0, return m and stop; otherwise go to Step 2
Step 2: Divide m by n and assign the value of the remainder to r
Step 3: Assign the value of n to m and the value of r to n. Go to Step 1.

```
while n ≠ 0 do
    r ← m mod n
    m← n
    n ← r
return m
```

Algorithm Design and Analysis



Algorithm Design Techniques

- Brute force
- Decrease and conquer
- Divide and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy Approach
- Dynamic Programming
- Iterative Improvement

Analysis of Algorithm

- How good is an algorithm?
  - time efficiency
  - space efficiency
- Does there exist a better algorithm?
  - lower bounds
  - optimality

Fundamental data structures

- Collection
  - Array
  - Vector
  - Linked list
  - String
- Stack
- Queue
- Priority queue
- Graph
- Tree
- Set
- dictionary

Important Problem Types

- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems