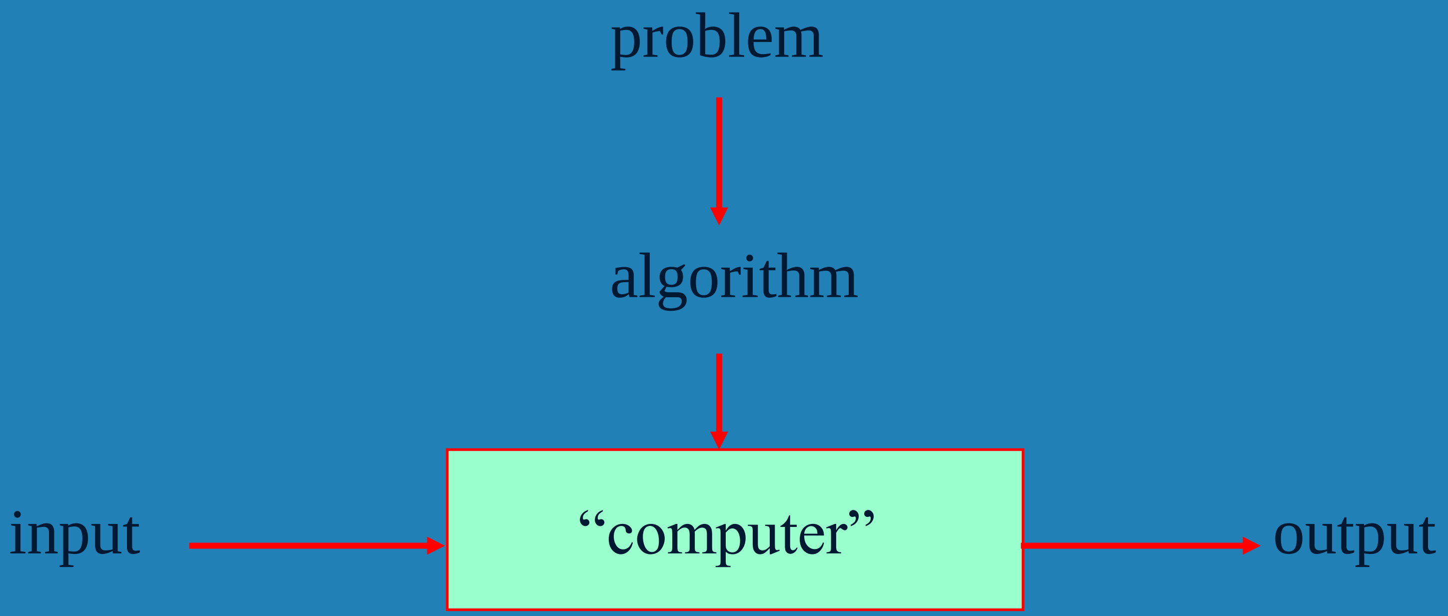
1.1 What is an algorithm?

An *algorithm* is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Two descriptions of 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 fo 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 $n \neq 0$ **do**

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

Other methods for computing gcd(m,n)

Consecutive integer checking algorithm

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

Other methods for gcd(m,n) [cont.]

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)

Is this an algorithm?

Example:

$60 = 2 \cdot 2 \cdot 3 \cdot 5$

$24 = 2 \cdot 2 \cdot 2 \cdot 3$

$\mathrm{gcd}(60, 24) = 2 \cdot 2 \cdot 3 = 12$

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ **to** n **do** $A[p] \leftarrow p$

for $p \leftarrow 2$ **to** $\lfloor \sqrt{n} \rfloor$ **do**

  if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

    $j \leftarrow p * p$

    while $j \leq n$ **do**

      $A[j] \leftarrow 0$ //mark element as eliminated

      $j \leftarrow j + p$

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

1.2 Algorithm design and analysis process

- **Understanding the Problem**
- **Ascertaining the Capabilities of the Computational Device**
  - **Sequential Computer Architecture: Random-access machine (RAM) Model → *Sequential algorithms***
  - **Parallel Computer Architecture → *parallel algorithms***
- **Choosing between Exact and Approximate Problem Solving**

Algorithm design and analysis process (Cont.)

- Selecting Algorithm Design Techniques
- Designing an Algorithm and Data Structures
- Methods of Specifying an Algorithm
  - Pseudocode
  - flowchart
- Proving an Algorithm's Correctness
- Coding an Algorithm

Why study algorithms?

- Theoretical importance
  - the core of computer science
- Practical importance
  - A practitioner's toolkit of known algorithms
  - Framework for designing and analyzing algorithms for new problems

Two main issues related to algorithms

- How to design algorithms

- How to analyze algorithm efficiency

Algorithm design techniques/strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound

Analysis of algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency

- Does there exist a better algorithm?
  - lower bounds
  - optimality

1.3 Important problem types

- Sorting
- Searching
- String processing

  Example: searching for a given word in a text
- Graph problems

  Examples: the traveling salesman problem and the graph-coloring problem
- Combinatorial problems

  To find a combinatorial object—such as a permutation, a combination, or a subset—that satisfies certain constraints

Important problem types (Cont.)

- Geometric problems

  Examples: the closest-pair problem and the convex-hull problem

- Numerical problems

  Examples: solving equations and systems of equations, computing definite integrals, evaluating functions

1.4 Fundamental data structures

- list
  - array
  - linked list
  - string
- stack
- queue
- priority queue
- graph
- tree
- set and dictionary