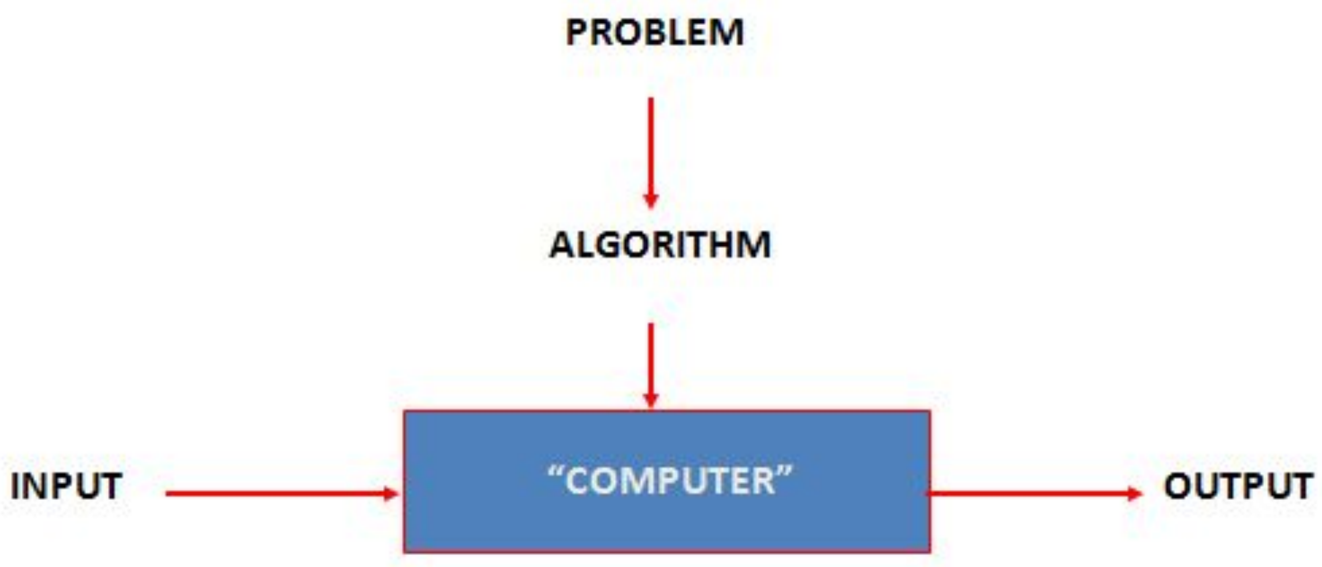
What is an algorithm?

- 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

- It should produce the output in finite amount of time
- It should be independent of programming language
- Every statement should e unambiguous
- Every two steps in an algorithm is related to each other otherwise the algorithm may stop without producing output.

- **Non-ambiguity**
- **Independent of Programming Language**
- **Range of inputs**
- **The same algorithm can be represented in different ways**
- **Several algorithms for solving the same problem**
- **Algorithm for same problem can be based on very different ideas and can solve the problem with different speeds.**

Notion of algorithm



NOTION OF AN ALGORITHM

What is an algorithm?

- Recipe, process, method, technique, procedure, routine,… with following requirements:

1. Finiteness
   - terminates after a finite number of steps
2. Definiteness
   - rigorously and unambiguously specified
3. Input
   - valid inputs are clearly specified
4. Output
   - can be proved to produce the correct output given a valid input
5. Effectiveness
   - steps are sufficiently simple and basic

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

Formative Assessment

1. A _______________ is a sequence of non ambiguous instructions for solving a problem in a finite amount of time.

A) Algorithm

B) Pseudo code

C) Flowchart

D) Mind map

2. An _____________ to an algorithm specifies an i*nstance* of the problem the algorithm solves.

A) Input

B) Output

C) Both input and output

D) Function

Historical Perspective

- Euclid's algorithm for finding the greatest common divisor

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,

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

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)

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 $\lfloor n \rfloor$ 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

Two main issues related to algorithms

- How to design algorithms
- How to analyze algorithm efficiency

Analysis of algorithms

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

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