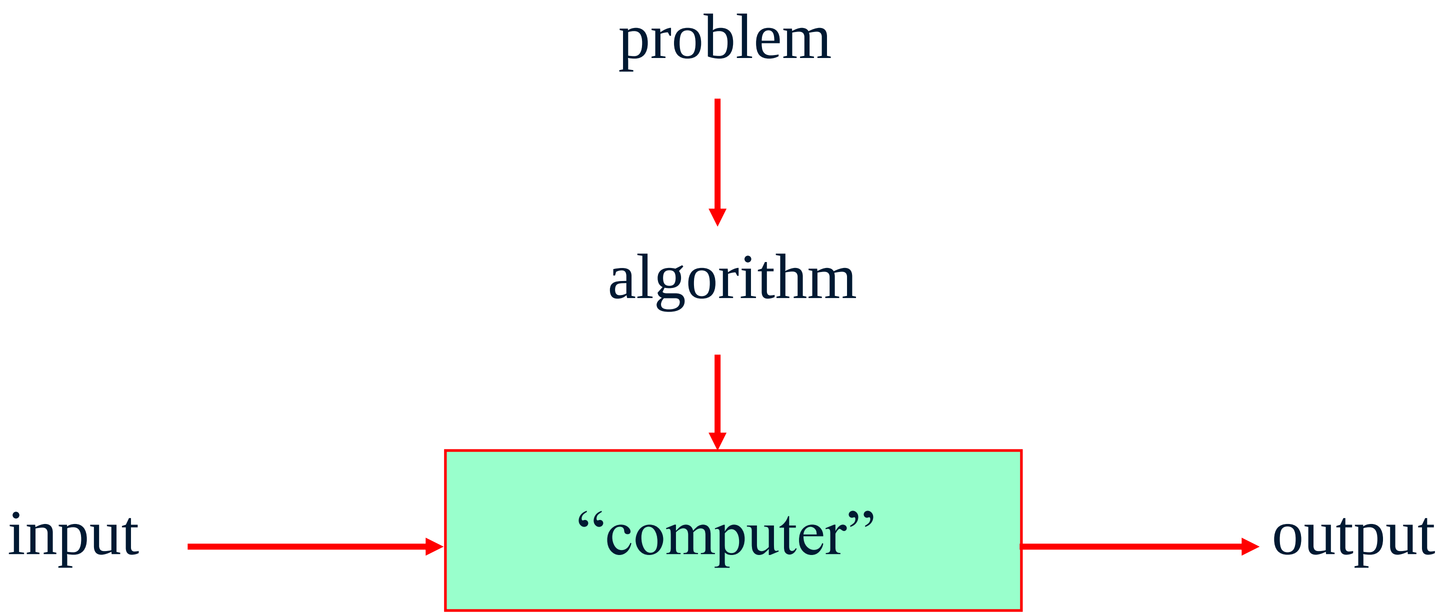
CSE408
Fundamentals of Algorithms

Lecture #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.



Algorithm

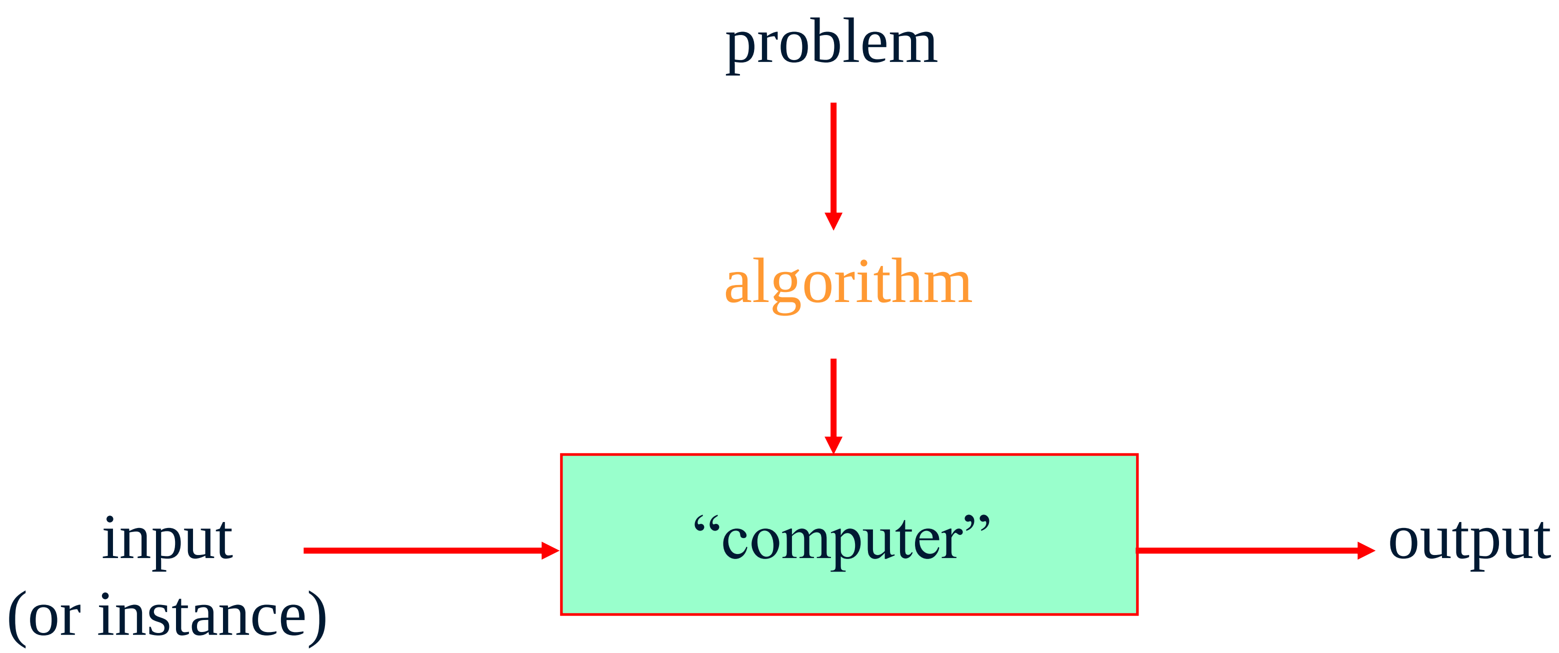
- An *algorithm* is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

Historical Perspective

- Euclid's algorithm for finding the greatest common divisor

- Muhammad ibn Musa al-Khwarizmi – 9^{th} century mathematician
  www.lib.virginia.edu/science/parshall/khwariz.html

Notion of algorithm and problem



(different from a conventional solution)

Example of computational problem: sorting

- Statement of problem:
  - *Input:* A sequence of n numbers $<a_1, a_2, \ldots, a_n>$
  - *Output:* A reordering of the input sequence $<a'_1, a'_2, \ldots, a'_n>$ so that $a'_i \le a'_j$ whenever $i < j$
- Instance: The sequence <5, 3, 2, 8, 3>
- Algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - (many others)

Selection Sort

- Input: array $a[1], \cdots, a[n]$
- Output: array a sorted in non-decreasing order
- Algorithm:

for i=1 to n
    swap $a[i]$ with smallest of $a[i], \cdots, a[n]$

Is this unambiguous? Effective?

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Primality testing
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi
- Program termination

Basic Issues Related to Algorithms

- How to design algorithms
- How to express algorithms
- Proving correctness
- Efficiency (or complexity) analysis
  - Theoretical analysis
  - Empirical analysis
- Optimality

Algorithm design strategies

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

Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency

- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

What is an algorithm?

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

1. Finiteness
   - terminates after a finite number of steps
2. Definiteness
   - rigorously and unambiguously specified
3. Clearly specified input
   - valid inputs are clearly specified
4. Clearly specified/expected 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

Example: Google’s PageRank Technology

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 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 $n \neq 0$ 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

Is this slower than Euclid's algorithm?
How much slower?

O(n), if n <= m , vs O(log n)

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?

How efficient is it?

Time complexity: O(sqrt(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 n do

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

   $j \leftarrow p * p$

   while $j \leq n$ do

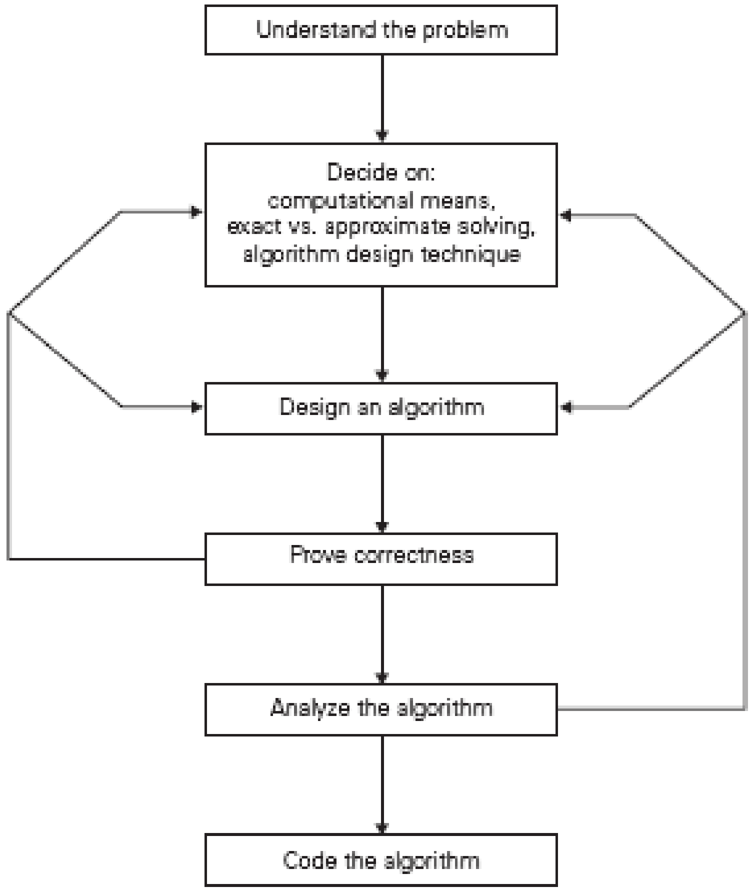
    $A[j] \leftarrow 0$ //mark element as eliminated

    $j \leftarrow j + p$

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

Time complexity: O(n)

Fundamentals of Algorithmic Problem Solving



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
  - correctness ignored in this course

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

Important problem types

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

Sorting (I)

- Rearrange the items of a given list in ascending order.
  - Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  - Output: A reordering $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.
- Why sorting?
  - Help searching
  - Algorithms often use sorting as a key subroutine.
- Sorting key
  - A specially chosen piece of information used to guide sorting. E.g., sort student records by names.

Sorting (II)

- Examples of sorting algorithms
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - Heap sort …
- Evaluate sorting algorithm complexity: the number of key comparisons.
- Two properties
  - Stability: A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input.
  - In place : A sorting algorithm is in place if it does not require extra memory, except, possibly for a few memory units.

Selection Sort

```
Algorithm SelectionSort(A[0..n-1])
//The algorithm sorts a given array by selection sort
//Input: An array A[0..n-1] of orderable elements
//Output: Array A[0..n-1] sorted in ascending order
for i ← 0 to n – 2 do
    min ← i
    for j ← i + 1 to n – 1 do
        if A[j] < A[min]
            min ← j
    swap A[i] and A[min]
```

Searching

- Find a given value, called a search key, in a given set.
- Examples of searching algorithms
  - Sequential search
  - Binary search …

Time: O(log n)

String Processing

- A string is a sequence of characters from an alphabet.
- Text strings: letters, numbers, and special characters.
- String matching: searching for a given word/pattern in a text.

Thank You !!!