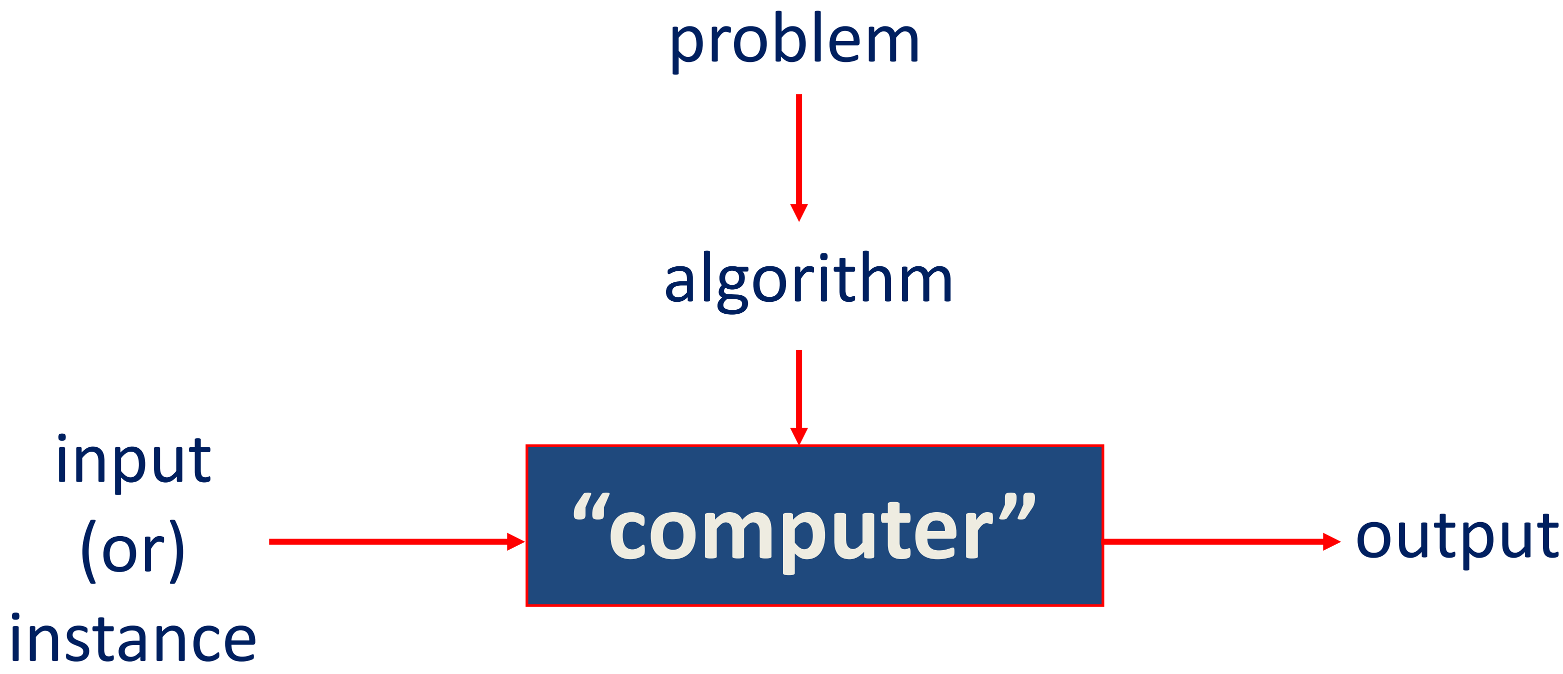
UNIT I INTRODUCTION

Introduction to Algorithms – Fundamentals of Algorithmic Problem Solving – Fundamentals of the Analysis of Algorithmic Efficiency – Analysis Framework – Asymptotic Notations and Basic Efficiency Classes – Mathematical Analysis of Recursive Algorithms – Mathematical Analysis of Non-recursive Algorithms

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.

- Can be represented various forms
- Unambiguity/clearness
- Effectiveness
- Finiteness/termination
- Correctness

Notion of algorithm and problem



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 $a'_i \leq a'_j$ whenever $i < j$
- **Instance:** The sequence **<5, 3, 2, 8, 3>**
- **Algorithms:**
  - Selection sort
  - Insertion sort
  - Merge sort
  - (many others)

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Traveling salesman problem
- Knapsack problem
- Towers of Hanoi

Some of these problems don’t have efficient algorithms, or algorithms at all!

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

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:

- **Finiteness**
  - terminates after a finite number of steps
- **Definiteness**
  - rigorously and unambiguously specified
- **Clearly specified input**
  - valid inputs are clearly specified
- **Clearly specified/expected output**
  - can be proved to produce the correct output given a valid input
- **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

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

Find GCF or GCD using the Euclidean Algorithm

Example:

Find GCD of 12 and 30

$30 \div 12 = 2$ remainder 6

$12 \div \boxed{6} = 2$ remainder 0

GCD

The GCD of 12 and 30 is 6

Find GCD of 123 and 36

$123 \div 36 = 3$ remainder 15

$36 \div 15 = 2$ remainder 6

$15 \div 6 = 2$ remainder 3

$6 \div \boxed{3} = 2$ remainder 0

GCD

The GCD of 123 and 36 is 3

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)

Consecutive Integer Checking Algorithm

- Example: gcd(10,6) = 2

t	m % t	n % t
6	10 % 6 = 4	
5	10 % 5 = 0	6 % 5 = 1
4	10 % 4 = 2	
3	10 % 3 = 1	
2	10 % 2 = 0	6 % 2 = 0

2 is the GCD, since m % t and n % t are zero.

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

Time complexity: sqrt(n)

Sieve of Eratosthenes

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

$$36 = 2 \times 2 \times 3 \times 3$$
$$60 = 2 \times 2 \times 3 \times 5$$

GCD = Multiplication of common factors

$$= 2 \times 2 \times 3$$
$$= 12$$

Thank You