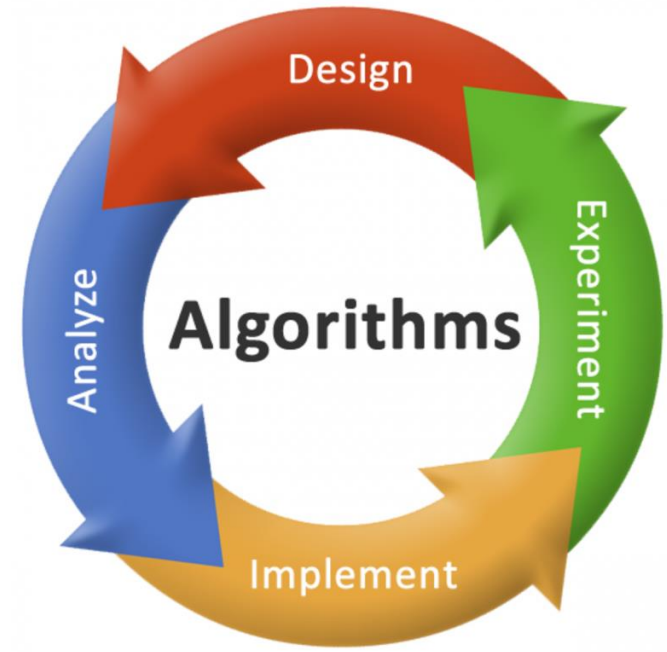


# DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 0 & 1 :

Introduction



# Overview

- Syllabus review
- Evaluation policy
- What is an algorithm?
- Why study algorithms?
- Algorithm requirements
- An example problem and the possible algorithms
- Algorithm design and analysis process
- Important problem types

# DSE 2256 DAA [3 + 1 = 4 credits] : Detailed Syllabus

**Introduction to DAA :** What is Algorithm, Fundamentals of Algorithms, Important Problem Types.

**Fundamentals of Analysis of Algorithms:** Analysis Framework: Asymptotic Notations and Basic Efficiency Classes, Mathematical Analysis of Non recursive and Recursive Algorithms.

**Brute force Techniques:** Selection sort, Bubble sort, Sequential Search, Brute Force String Matching, Exhaustive Search.

**Divide and Conquer Techniques:** Merge Sort, Quick Sort, Binary Search, Binary Tree Traversals, Multiplication of large integers , Strassen's Matrix Multiplication.

**Decrease and Conquer Techniques:** Insertion Sort, Depth First Search, Breadth First Search, Topological Sorting.

**Transform and Conquer Techniques:** Presorting, Balanced Search Trees, Heapsort, Problem reduction.

# DSE 2256 DAA [3 + 1 = 4 credits] : Detailed Syllabus (contd.)

**Space and Time trade-offs:** Sorting by counting, Input Enhancement in String Matching (Horspool algorithm, Boyer-Moore algorithm), Hashing.

**Dynamic Programming:** Computing a Binomial Coefficient, Warshall's and Floyd's Algorithms, The Knapsack Problem and Memory functions.

**Greedy Techniques:** Prim's, Kruskal's and Dijkstra's Algorithm, Huffman Trees.

**Limitations of algorithmic power:** P, NP, and NP-complete Problems.

**Coping with the limitations of algorithmic power:** Backtracking, n-Queens problem, Hamiltonian Circuit Problem, Subset-Sum Problem.

**Branch and Bound:** Assignment Problem, Knapsack Problem, Travelling Salesman Problem.

# DSE 2256 DAA : References

1. **Anany Levitin**, *Introduction to the Design and Analysis of Algorithms*, (3e), Pearson Education, 2011.
2. **Ellis Horowitz and Sartaj Sahni**, *Computer Algorithms/C++*, (2e), University Press, 2007.
3. **Thomas H. Cormen, Charles E. Leiserson, Ronal L, Rivest, Clifford Stein**, *Introduction to Algorithms*, (2e), PHI, 2006.
4. **MIT OpenCourseWare**: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-spring-2020/lecture-videos/>
5. **NPTEL Video Lectures**: <https://nptel.ac.in/courses/106/106/106106131/>

# DSE 2256 DAA : Evaluation Policy (Tentative)

**Assignment#1:** 5 marks

**Assignment#2:** 5 marks

**Assignment#3:** 5 marks

**Assignment#4:** 5 marks

$4 * 5 = 20$  marks

**Sessional Test#1:** 15 marks

**Sessional Test#2:** 15 marks

$2 * 15 = 30$  marks

**Internal Max. Marks = 50**

**End Semester Examination: 50 marks**

# DSE 2256 DAA : Prerequisites

- Knowledge about fundamental data structures
- Recurrence relations
- Necessary summation formulas
- Calculus to some extent
- Basic programming knowledge

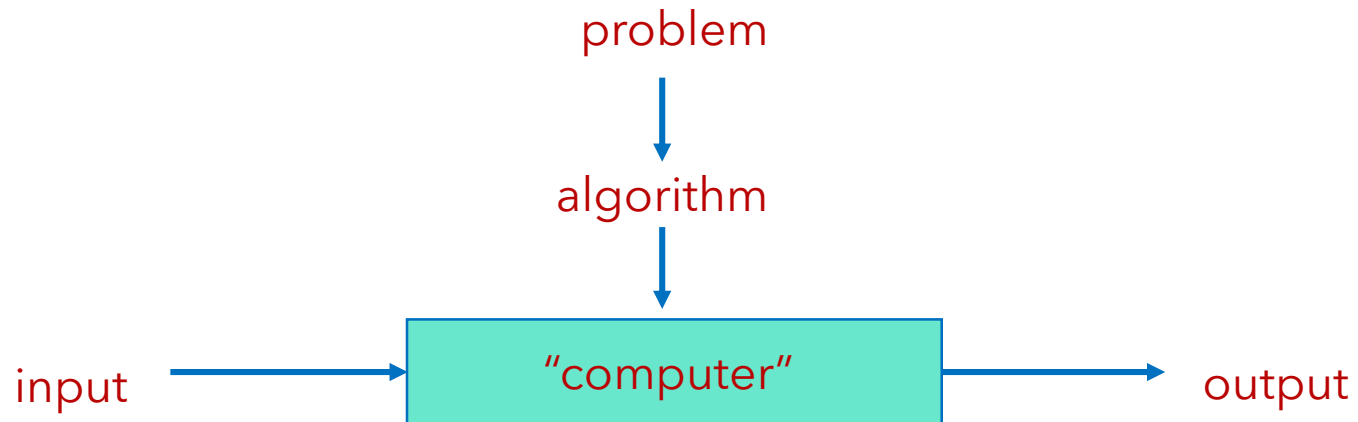


DSE 2256 Design &amp; Analysis of Algorithms



# What is an Algorithm?

- An **algorithm** is a sequence of **unambiguous instructions**:
    - ❖ For solving a problem.
- OR**
- ❖ For obtaining a **required output** for any **legitimate input** in a **finite amount of time**.



# Why study algorithms?

- **Practical importance**

- A practitioner's toolkit of known algorithms
- Framework for designing and analyzing algorithms for new problems

- **Theoretical importance**

- The core of computer science

The study of algorithms is called **algorithmics**.

# Algorithm requirements

- **Finiteness**
  - Terminates after a finite number of steps
- **Definiteness**
  - Each step must be clear and unambiguous
- **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

## Remember!!

- The same algorithm can be represented in different ways.
- Several algorithms for solving the same problem may exist.

# Example : GCD of two numbers

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

## Examples:

1. If  $m = 60$  and  $n = 24$ ,

$$\text{gcd}(60,24) = 12$$

2. If  $m = 60$  and  $n = 0$ ,

$$\text{gcd}(60,0) = 60$$

3. If  $m = 0$  and  $n = 0$ ,

$$\text{gcd}(0,0) = ?$$

## Question:

1. What if m and n are prime numbers ?
2. What if m and n are co-prime or relatively prime numbers?

# Euclid's Algorithm :

**The logic:** Euclid's algorithm is based on repeated application of following equality until the second number becomes 0 :

$$\text{gcd}(m,n) = \text{gcd}(n, m \bmod n)$$

## Example:

$$\text{gcd}(60,24) = ?$$

$$\begin{aligned}\text{gcd}(60,24) &= \text{gcd}(24, 60 \bmod 24) \\ &= \text{gcd}(24,12) \\ &= \text{gcd}(12, 24 \bmod 12) \\ &= \text{gcd}(12,0) \\ &= \mathbf{12}\end{aligned}$$

# Two descriptions of Euclid's algorithm

## #1

**Euclid's algorithm** for computing  $\text{gcd}(m,n)$

**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**.

## #2

**Algorithm** Euclid( $m,n$ )

//Computes  $\text{gcd}(m,n)$  by Euclid's algorithm

// Input: two nonnegative, not both zero integers  $m$  and  $n$

//Output: Greatest common divisor of  $m$  and  $n$

while  $n \neq 0$  do

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return  $m$

# Other algorithms to compute GCD – I

**Consecutive integer checking algorithm** for computing  $\text{gcd}(m, n)$

**Step 1** : Assign the value of  $\min\{m, n\}$  to  $t$ .

**Step 2** : **Divide  $m$  by  $t$** . If the remainder of this division is 0, go to **Step 3**; otherwise, go to **Step 4**.

**Step 3** : **Divide  $n$  by  $t$** . If the remainder of this division is 0, return the value of  $t$  as the answer and stop; otherwise, proceed to **Step 4**.

**Step 4** : **Decrease the value of  $t$  by 1**. Go to **Step 2**.

Is this slower than Euclid's algorithm?

# Other algorithms to compute GCD - II

**Middle-school procedure** for computing  $\text{gcd}(m, n)$

**Step 1** : Find the prime factors of  $m$ .

**Step 2** : Find the prime factors of  $n$ .

**Step 3** : Identify all the common factors in the two prime expansions found in [Step 1](#) and [Step 2](#) (If  $p$  is a common factor occurring  $p_m$  and  $p_n$  times in  $m$  and  $n$ , respectively, it should be repeated  $\min\{p_m, p_n\}$  times).

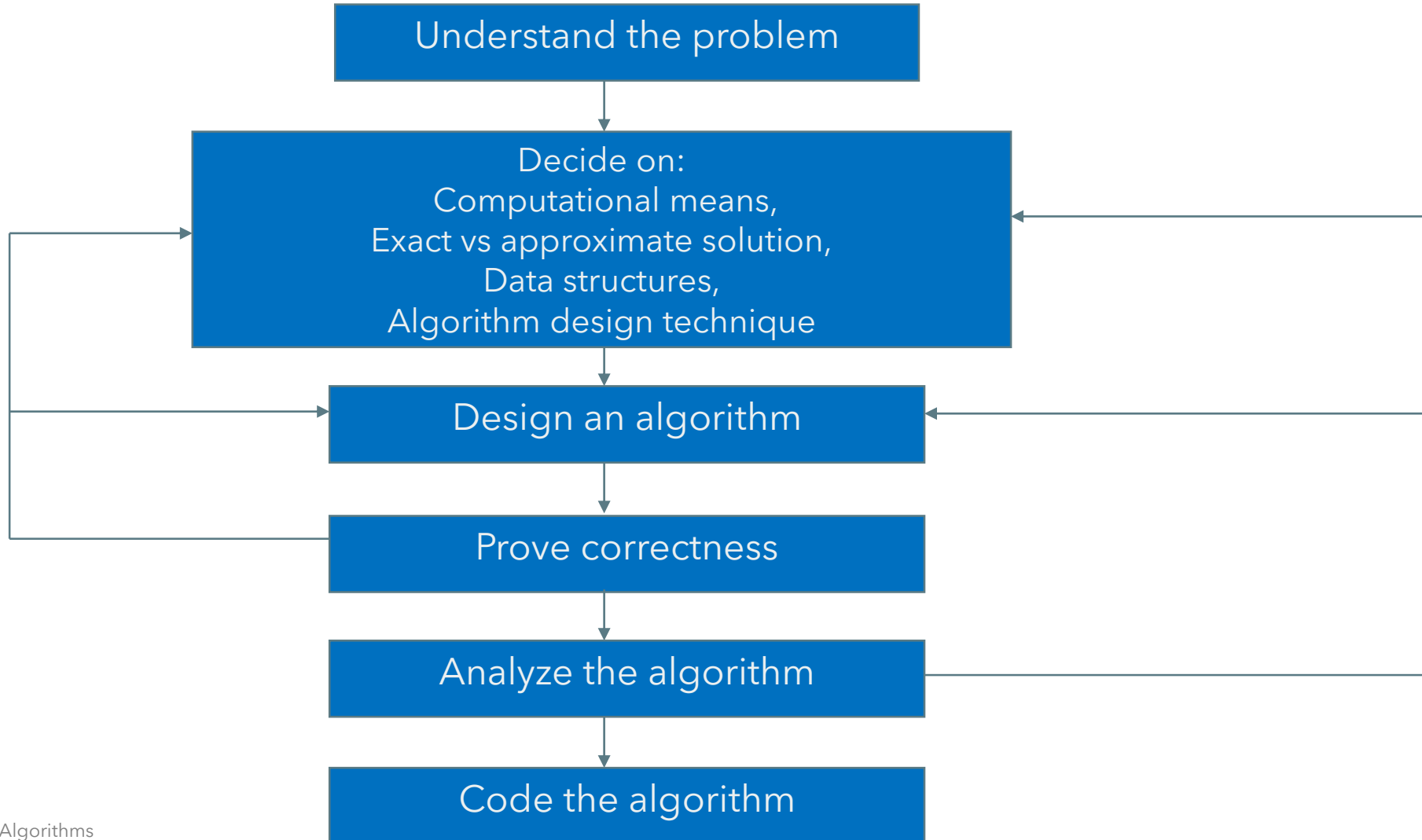
**Step 4** : Compute the product of all the common factors and return it as the GCD of the numbers given.



# 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 and analysis process



# Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency
- Other Characteristics
  - Simplicity
  - Generality
- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

# Important problem types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

# Important problem types I

- **Sorting**
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

8 5 3 1 4 7 9

# Important problem types II

- Sorting
- **Searching**
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

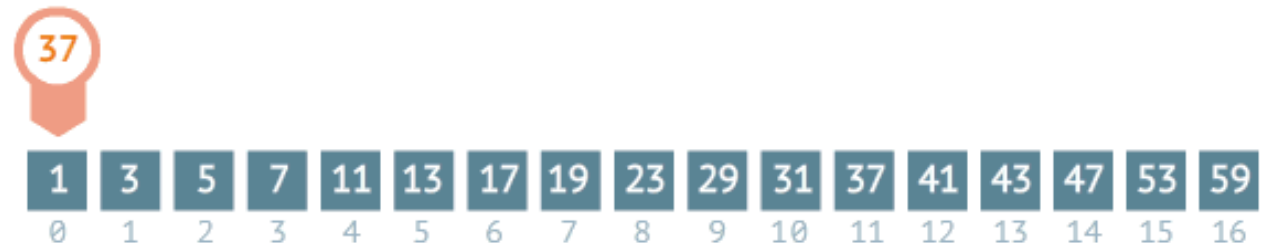
Binary search

steps: 0



Sequential search

steps: 0



Courtesy: [www.penjee.com](http://www.penjee.com)

# Important problem types III

- Sorting
- Searching
- **String processing**
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

The text "HELLO WORLD" is displayed in a stylized font where each letter is contained within its own green rectangular block. The blocks are arranged in a single horizontal row with a small gap between the "O" and "W" blocks. The letters are white and uppercase.

Courtesy: [www.raywenderlich.com](http://www.raywenderlich.com)

# Important problem types IV

- Sorting
- Searching
- String processing
- **Graph problems**
- Combinatorial problems
- Geometric problems
- Numerical problems

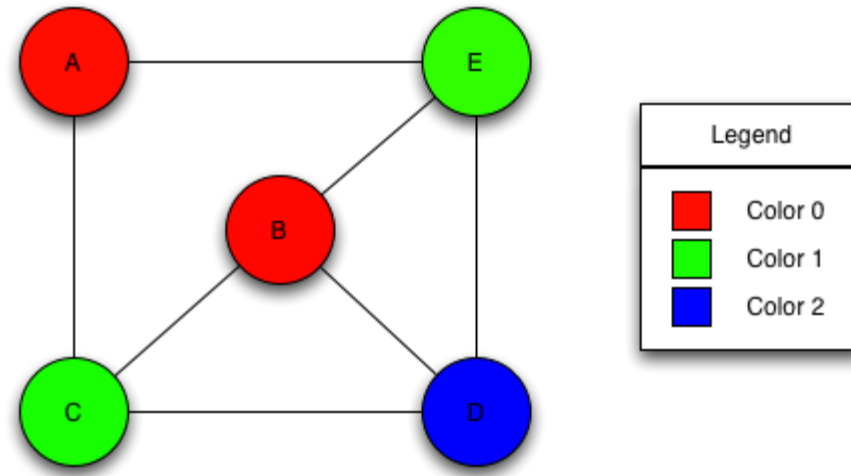


Courtesy: [www.manipal.pure.elsevier.com](http://www.manipal.pure.elsevier.com)



# Important problem types V

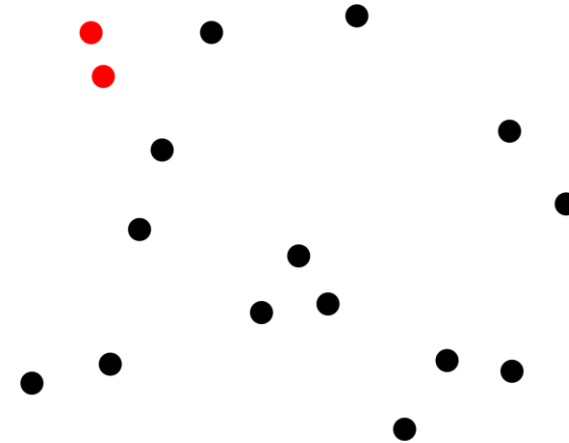
- Sorting
- Searching
- String processing
- Graph problems
- **Combinatorial problems**
- Geometric problems
- Numerical problems



Courtesy: [www.boost.org](http://www.boost.org)

# Important problem types VI

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- **Geometric problems**
- Numerical problems



# Important problem types VII

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- **Numerical problems**

$$e^x \approx 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}.$$

# Summary

- An algorithm is sequence of **unambiguous instructions** for obtaining a required output from any **legitimate input** in a **finite amount of time**.
- An algorithm can be written using **English like language** or using **pseudocodes** or **a mix of both**.
- The **same algorithm** can be represented in **different ways** and **several algorithms** for the **same problem** may exist.
- The **best algorithm** for a given problem **must generate legitimate outputs** for **all specified range of inputs**, must **utilize less memory space** and **is faster than all other algorithms** for the same problem.

# Thank you!

## Any queries?