# Analisis Algoritma dan Kompleksitas

Nur Rokhman

# Design & Analysis of Algorithms

- **Algorithm analysis**

  Analysis of resource usage of given algorithms (time , space)

- **Efficient algorithms**

  Algorithms that make an efficient usage of resources

- **Algorithm design**

  Methods for designing efficient algorithms

# Design & Analysis of Algorithms



## Why study this subject?

- Efficient algorithms lead to efficient programs.
- Efficient programs sell better.
- Efficient programs make better use of hardware.
- Programmers who write efficient programs are preferred.

# Objectives

- To gain experiences in fundamental techniques used for algorithm analysis and the main methodologies used for the design of efficient algorithms.
- To study the most important computer algorithms of current practical use.

# Contents



- **Analysis of Iterative and Recursive Algorithms**

1. **Brute Force Algorithms**
2. **Recursive Algorithms**

- **Major Algorithm Design Methodologies**

1. **Transform & Conquer Algorithms**
2. **Divide & Conquer Algorithms**
3. **Greedy Algorithms**
4. **Dynamic Programming**
5. **Backtracking Algorithms**
6. **Graph Algorithms**
7. **Branch & Bound**
8. **Other Strategies (Heuristics, String & Numerical Algorithms)**

# Course Outcomes



**After completing the course, students should be able to:**

1. Determine the time and space complexity of simple algorithms.
2. Use big *O*, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
3. Recognize the difference between mathematical modeling and empirical analysis of algorithms, and the difference between deterministic and randomized algorithms.
4. Deduce recurrence relations that describe the time complexity of recursively defined algorithms and work out their particular and general solutions.

# Course Outcomes



5. Practice the main algorithm design strategies of Brute Force, Divide & Conquer, Greedy methods, Dynamic Programming, Backtracking and Branch & Bound and implement examples of each.
6. Implement the most common sorting and searching algorithms and perform their complexity analysis.
7. Solve problems using the fundamental graph algorithms including DFS, BFS, SSSP and APSP, transitive closure, topological sort, and the minimum spanning tree algorithms.
8. Evaluate, select and implement algorithms in programming context.

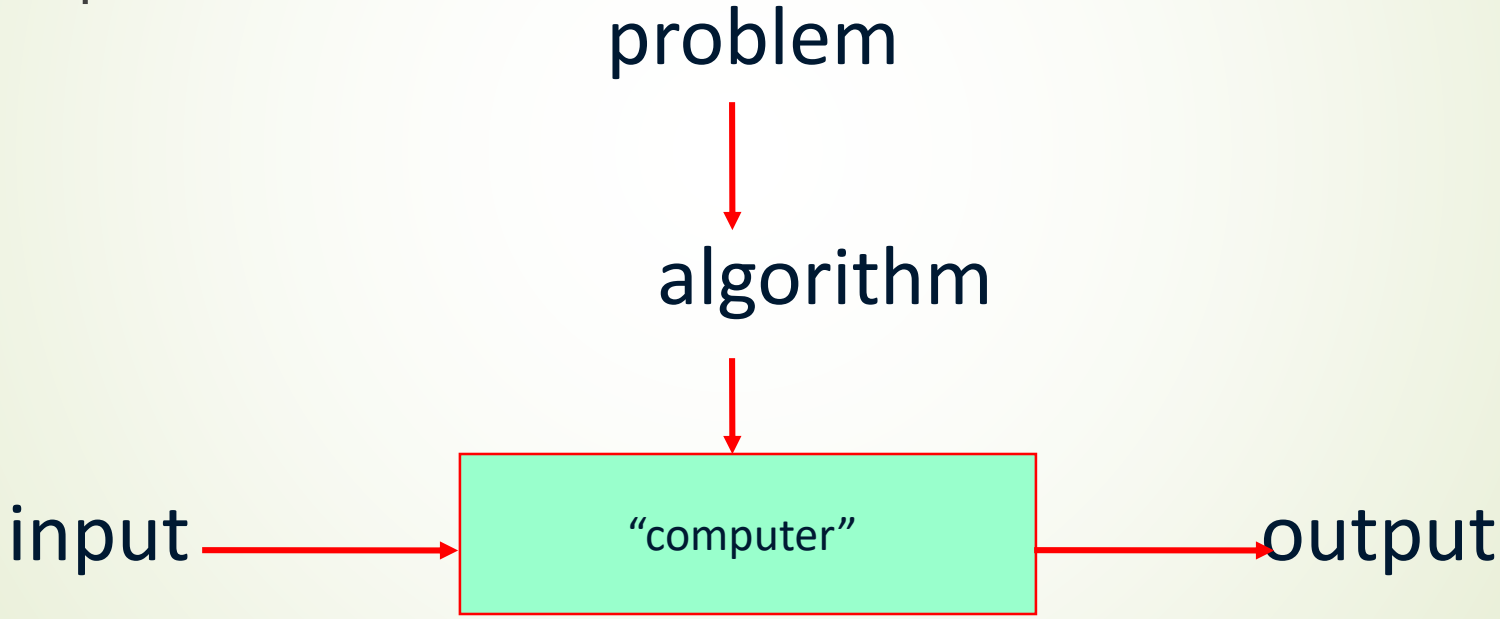
# UNIT – I - Introduction

Notion of an Algorithm – Fundamentals of Algorithmic Problem Solving – Important Problem Types – Fundamentals of the Analysis of Algorithm Efficiency – Analysis Framework – Asymptotic Notations and its properties – Mathematical analysis for Recursive and Non-recursive algorithms.

# What is an algorithm?



An *algorithm* is a list of steps (sequence of unambiguous instructions ) for solving a problem that transforms the input into the output.



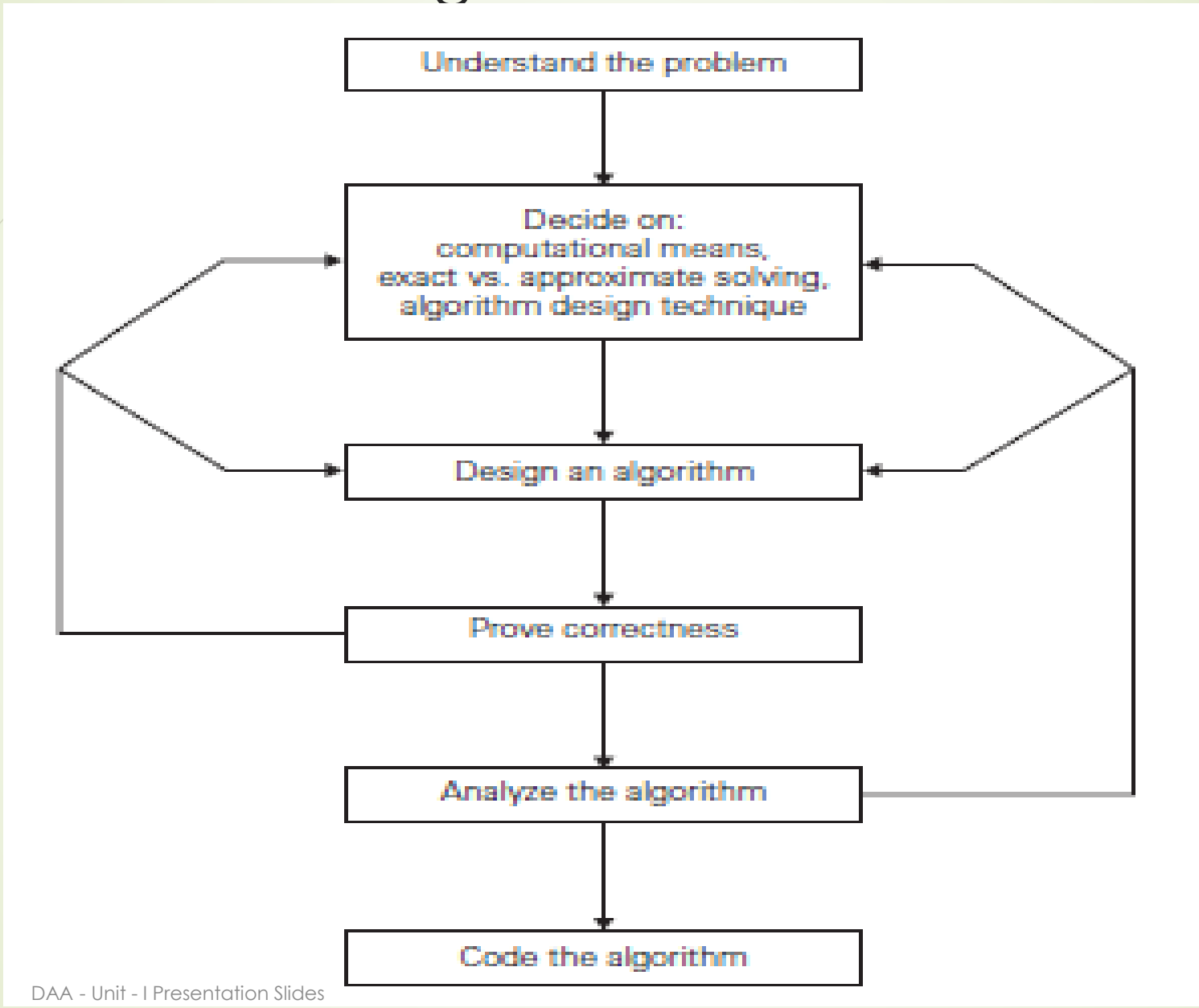

# Difference between Algorithm and Program



| S. No | Algorithm | Program |
|---|---|---|
| 1 | Algorithm is finite | Program need not to be finite |
| 2 | Algorithm is written using natural language or algorithmic language | Programs are written using a specific programming language |

# Fundamentals of Algorithm and Problem Solving






# Problem Solving Techniques



1. Understand the problem or Review the Specifications.
2. Plan the logic
3. a) (Informal Design)
   i. List major tasks
   ii. List subtasks,sub-subtasks & so on
   b) (Formal Design)
   i. Create formal design from task lists
   ii. Desk check design
4. Writing an algorithm 5. Flowcharting 6. Coding
7. Translate the program into machine language
8. Test the program
   i. If necessary debug the program
10. Documentation
11. Put the program into production. If necessary maintain the program.

# Example of computational problem: sorting



- Arranging data in a specific order (increasing or decreasing) is called sorting. The data may be numerical data or alphabetical data.

  $A_1 \leq A_2 \leq A_3 \leq \ldots\ldots \leq A_n$ or

  $A_n \geq A_{n-1} \geq A_{n-2} \geq \ldots\ldots \geq A_1 \geq A_0$

- Internal Sorting

  Here, all data are held in primary memory during the sorting process.

- External Sorting

  Here, it uses primary memory for the data currently being sorted and uses secondary storage for string data.

# Types of Sorting



- Internal Sorting
  - Insertion (Insertion sort, Address Calculation sort, Shell sort)
  - Selection (Selection sort, Heap sort)
  - Exchange (Bubble sort, Quick sort, Radix sort)
- External Sorting
  - Natural sort
  - Merge sort
  - Multi-way merge sort
  - Balanced sort
  - Polyphase sort
-

# Selection Sort



Suppose A is an array which consists of '*n*' elements namely A[I], A[2], . . . , A[N]. The selection sort algorithm will works as follows.

1. Step 1: a. First find the location LOC of the smallest element in the list A[I], A[2], . . . , A[N] and put it in the first position.

   b. Interchange A[LOC] and A[1].

   c. Now, A[1] is sorted.

2. Step 2: a. Find the location of the second smallest element in the list A[2], . . . , A[N] and put it in the second position.

   b. Interchange A[LOC] and A[2].

   c. Now, A[1] and A[2] is sorted. Hence, A[I] ≤ A[2].

3. Step 3: a. Find the location of the third smallest element in the list A[3], . . . , A[N] and put it in the third position.

   b. Interchange A[LOC] and A[3].

   c. Now, A[1], A[2] and A[3] is sorted. Hence, A[I] ≤ A[2] ≤ A[3].

. . . . . .

N-1. Step N–1 : a. Find the location of the smallest element in the list A[A–1] and A[N].

b. Interchange A[LOC] and A[N–1] & put into the second last position.

c. Now, A[1], A[2], …..,A[N] is sorted. Hence, A[I] ≤ … ≤ A[N–1] ≤A[N].

| Pass | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| Initital | 44 | 55 | 12 | 42 | 94 | 06 | 18 | 67 |
| K=1,LOC=**6** | 06 | 55 | 12 | 42 | 94 | 44 | 18 | 67 |
| K=2,LOC=**3** | 06 | 12 | 55 | 42 | 94 | 44 | 18 | 67 |
| K=3,LOC=**7** | 06 | 12 | 18 | 42 | 94 | 44 | 55 | 67 |
| K=4,LOC=**4** | 06 | 12 | 18 | 42 | 94 | 44 | 55 | 67 |
| K=5,LOC=**6** | 06 | 12 | 18 | 42 | 44 | 94 | 55 | 67 |
| K=6,LOC=**7** | 06 | 12 | 18 | 42 | 44 | 55 | 94 | 67 |
| K=7,LOC=**8** | 06 | 12 | 18 | 42 | 44 | 55 | 67 | 94 |
| SORTED | 06 | 12 | 18 | 42 | 44 | 55 | 67 | 94 |

# 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

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

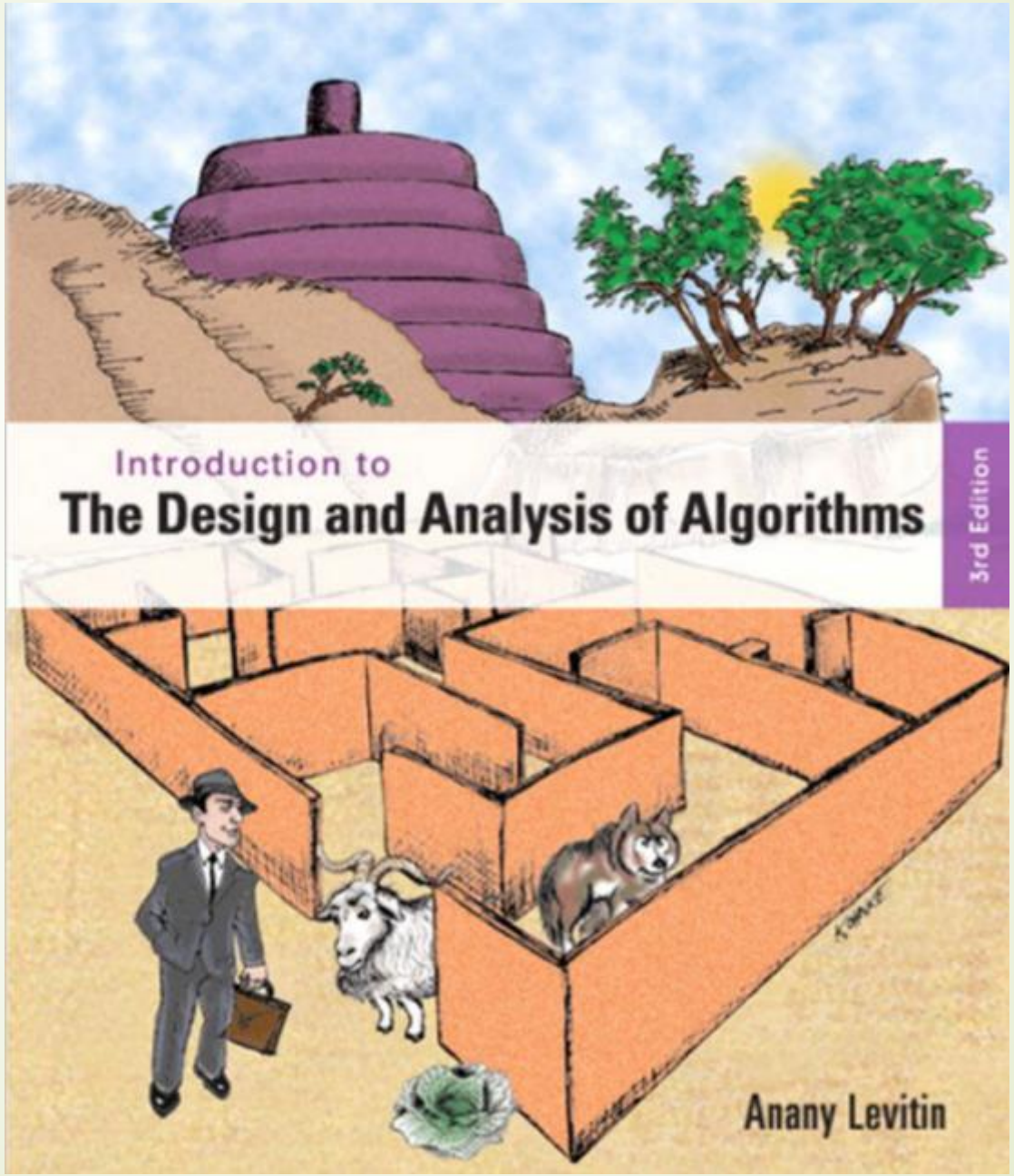


- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- **Greedy approach**
- **Dynamic programming**
- **Backtracking and branch-and-bound**
- **Space and time tradeoffs**

# PROPERTIES OF AN ALGORITHM



1. An algorithm takes zero or more inputs
2. An algorithm results in one or more outputs
3. All operations can be carried out in a finite amount of time
4. An algorithm should be efficient and flexible
5. It should use less memory space as much as possible
6. An algorithm must terminate after a finite number of steps.
7. Each step in the algorithm must be easily understood for some reading it


Introduction to
The Design and Analysis of Algorithms
3rd Edition
Anany Levitin