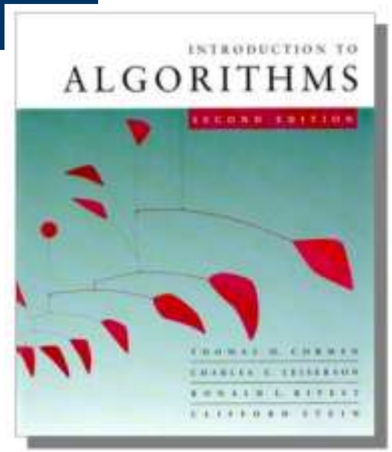
# Analysis and Design of Algorithms

By
Syed Bakhtawar Shah Abid
Lecturer in Computer Science

# Information

- Textbook
  - ***Introduction to Algorithms 2nd*** *,Cormen, Leiserson, Rivest and Stein*, The MIT Press, 2001.




- Others
  - ***Introduction to Design & Analysis Computer Algorithm 3rd,*** Sara Baase, Allen Van Gelder, Adison-Wesley, 2000.
  - ***Algorithms***, Richard Johnsonbaugh, Marcus Schaefer, Prentice Hall, 2004.
  - ***Introduction to The Design and Analysis of Algorithms 2nd Edition,*** Anany Levitin, Adison-Wesley, 2007.

# Course Objectives

- This course introduces students to the analysis and design of computer algorithms. Upon completion of this course, students will be able to do the following:
  - Analyze the asymptotic performance of algorithms.
  - Demonstrate a familiarity with major algorithms and data structures.
  - Apply important algorithmic design paradigms and methods of analysis.
  - Synthesize efficient algorithms in common engineering design situations.

# What is Algorithm?

- Algorithm
  - Is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
  - is thus a sequence of computational steps that transform the input into the output.
  - Is a tool for solving a well - specified computational problem.
  - Any special method of solving a certain kind of problem (Webster Dictionary)

# Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

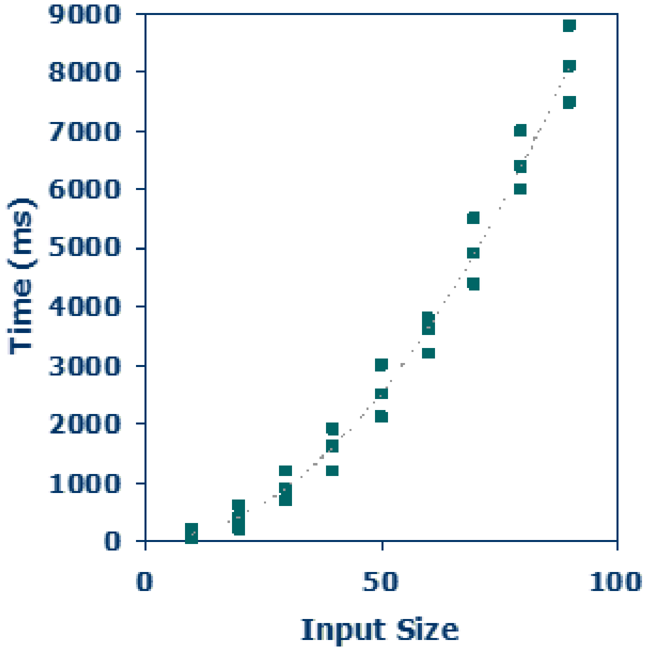
| **Algorithm** *arrayMax*($A$, $n$) | # operations |
|---|---|
| $currentMax \leftarrow A[0]$ | ? |
| **for** $i \leftarrow 1$ **to** $n - 1$ **do** | ? |
|     **if** $A[i] > currentMax$ **then** | ? |
|         $currentMax \leftarrow A[i]$ | ? |
| { increment counter $i$ } | ? |
| **return** *currentMax* | ? |
| Total | ? |

## What is a program?

- A program is the expression of an algorithm in a programming language
- A set of instructions which the computer will follow to solve a problem

# Assignment#1

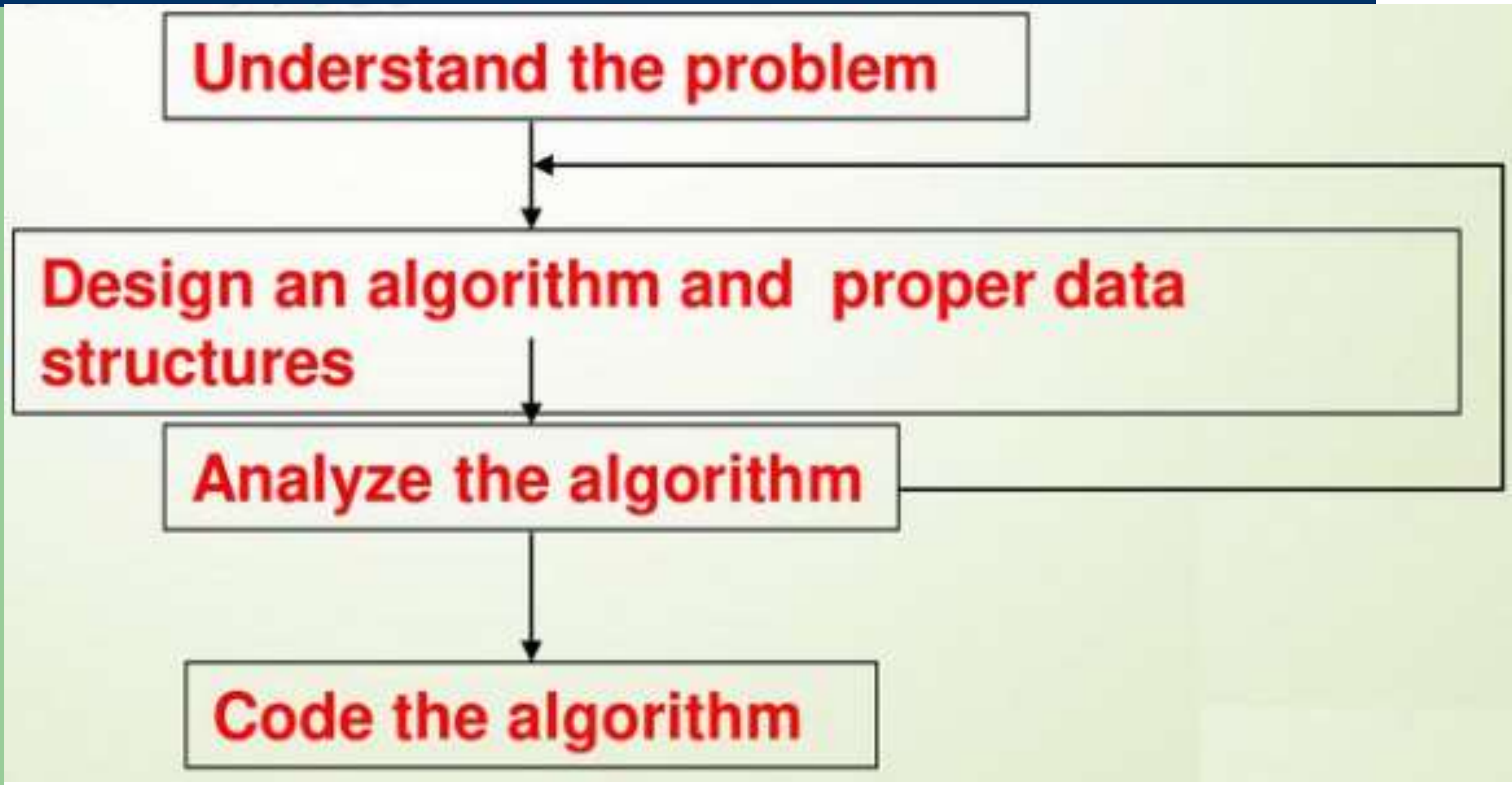
- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a function, like the built-in clock() function, to get an accurate measure of the actual running time
- Plot the results



# Design and Analysis

- Design
  - The design pertains to
    - The description of an algorithm at an abstract level by means of pseudocode and
    - Proof of correctness that is, the algorithm solves the given problem in all cases
- Analysis
  - The analysis deals with performance evaluation (complexity analysis)

# Algorithm development process



# Approaches to Algorithm Design

- Learn general approaches to algorithm design
  - Divide and conquer
  - Greedy method
  - Dynamic Programming
  - Basic Search and Traversal Technique
  - Graph Theory
  - Linear Programming
  - Approximation Algorithm
  - NP Problem

# Approaches to Algorithm Design

- Examine methods of analyzing algorithm correctness and efficiency
- Decide whether some problems have no solution in reasonable time
  - List all permutations of n objects (takes n! steps)
  - Travelling salesman problem
- Investigate memory usage as a different measure of efficiency

# Some Application

- Study problems these techniques can be applied to
  - sorting
  - data retrieval
  - network routing
  - Games
  - etc

# The study of Algorithm

- How to devise algorithms
- How to express algorithms
- How to validate algorithms
- How to analyze algorithms
- How to test a program

# Importance of Analysis

- Need to recognize limitations of various algorithms for solving a problem
- Need to understand relationship between problem size and running time
  - When is a running program not good enough?
- Need to learn how to analyze an algorithm's running time without coding it
- Need to learn techniques for writing more efficient code
- Need to recognize bottlenecks in code as well as which parts of code are easiest to optimize

## Why do we analyze about them?

- Understand their behavior, and
- Improve them. (Research)

# What do we analyze about them?

- Correctness
  - Does the input/output relation match algorithm requirement?
- Amount of work done (aka complexity)
  - Basic operations to do task
- Amount of space used
  - Memory used

# What do we analyze about them?

- Simplicity, clarity
  - Verification and implementation.
- Optimality
  - Is it impossible to do better?

# Complexity

- The complexity of an algorithm is simply the amount of work the algorithm performs to complete its task.

# What's more important than performance?

- Modularity
- Correctness
- Maintainability
- Functionality
- Robustness
- User-friendliness
- Programmer time
- Simplicity
- Extensibility
- Reliability

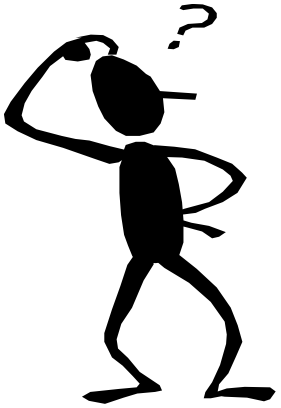
# The Selection Problem

- Problem: given a group of n numbers, determine the $k^{th}$ largest
- Algorithm 1
  - Store numbers in an array
  - Sort the array in descending order
  - Return the number in position k

# The Selection Problem

- Algorithm 2
  - Store first k numbers in an array
  - Sort the array in descending order
  - For each remaining number, if the number is larger than the $k^{th}$ number, insert the number in the correct position of the array
  - Return the number in position k

Which algorithm is better?



# Define Problem

- **Problem**:
  - Description of Input-Output relationship
- **Algorithm**:
  - A sequence of computational step that transform the input into the output.
- **Data Structure:**
  - An organized method of storing and retrieving data.
- **Our task:**
  - Given a problem, design a ***correct*** and ***good*** algorithm that solves it.

# Example Algorithm A

**Problem:** The input is a sequence of integers stored in array. Output the minimum.

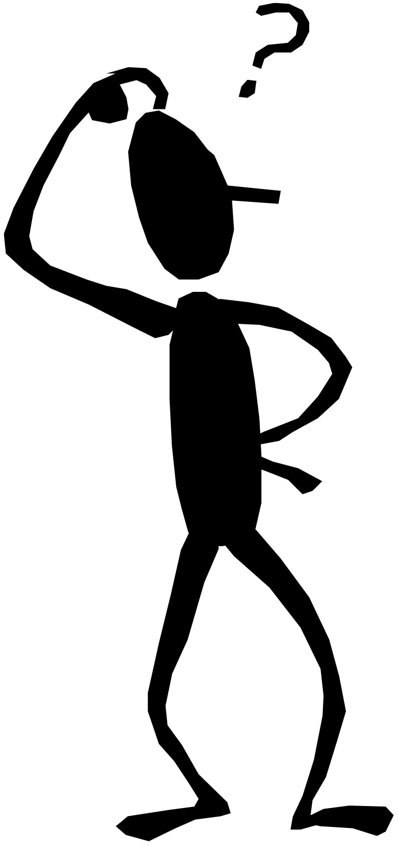
**Algorithm A**

$m \leftarrow a[1]$;
For $i \leftarrow 2$ to size of input;
    if $m > a[i]$ then $m \leftarrow a[i]$;
output $m$.

# Which algorithm is better?

**The algorithms are correct, but which is the best?**

- Measure the running time (number of operations needed).
- Measure the amount of memory used.
- Note that the running time of the algorithms increase as the size of the input increases.



# What do we need?

**Correctness:** Whether the algorithm computes the correct solution for **all** instances

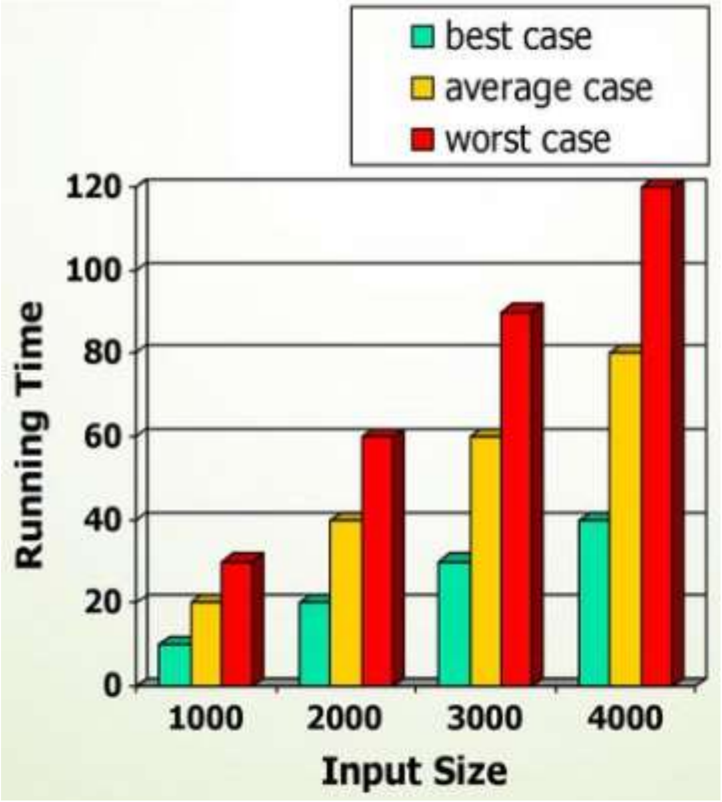
**Efficiency:** Resources needed by the algorithm

1. Time: Number of steps.
2. Space: amount of memory used.

Measurement "model": Worst case, Average case and Best case.

# Running Time

- Most algorithms transform input objects into output objects.
- The running time of an algorithm typically grows with the input size.
- Average case time is often difficult to determine.
- We focus on the worst case running time.
  - Easier to analyze
  - Crucial to applications such as games, finance and robotics



# What is Algorithm Analysis?

- How to estimate the time required for an algorithm
- Techniques that drastically reduce the running time of an algorithm
- A mathemactical framwork that more rigorously describes the running time of an algorithm

# Theoretical analysis of running time

- Theoretical analysis
  - Uses the pseudocode description of an algorithm rather than the actual implementation
  - Characterizes running time as a function of the input size n
  - Takes into account all possible inputs
  - Allows us to evaluate the speed of an algorithm independent of the hardware/software environment (Random Access Machine (RAM))

# Input Size

- Time and space complexity
  - This is generally a function of the input size
    - E.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - Etc

# Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call, most statements roughly require the same amount of time
    - y = m * x + b
    - c = 5 / 9 * (t - 32 )
    - z = f(x) + g(y)
- We can be more exact if need be

# Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?
    - Random (equally likely) inputs
    - Real-life inputs