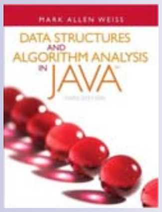
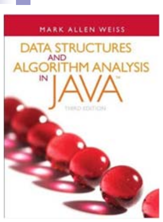



FLORIDA INTERNATIONAL UNIVERSITY

# Algorithm Analysis (I)

Dr. Antonio L. Bajuelos
FIU School of Computing & Information Sciences






## COP-3530 - Data Structures

**Module #1: Algorithm Analysis (part I)**

**Outline:**

- What is algorithm?
- How efficient is an algorithm?
- Performance vs Complexity
- T(n) - the relation of the number of operations to the problem size

## The first claim of this course…

"**Bad** programmers worry about the **code**. **Good** programmers worry about **data structures** and their relationships."

(by Linus Trovals, creator of **Linux** kernel)

3

## Some Terminology

- **Algorithm**
  - A high-level description of a piece-by-piece or step-by-step process.
- **Abstract Data Type (ADT)**
  - Formal (Mathematical) description/model of components of the data and the set of operations that are allowed.
- **Data structure**
  - A specific organization of data for implementing an ADT
- **Implementation**
  - A specific implementation of a data structure in a specific programming language.

4

## The Definition of an Algorithm

- Brief and concise definition:

An **algorithm** is a clearly specified set of simple instructions to be followed to solve a problem.

5

## The Definition of an Algorithm

- A more complete definition:

An **algorithm** is a well-ordered collection of clear and effectively computable operations that when executed produces a result and halts in a finite amount of time*.

*Schneider, M. and J. Gersting (1995), *An Invitation to Computer Science*, West Publishing Company, New York, NY, p. 9.

6

## The Definition of an Algorithm

- Some **relevant** points of this definition:
  - An algorithm is a **well-ordered** collection...
  - An algorithm has **clear** operations...
  - An algorithm has **effectively computable** operations...
  - An algorithm **produces a result**...
  - An algorithm **halt in a finite amount of time**

7

## Algorithm Efficiency

- **How efficient is an algorithm?**
  - **Efficiency** covers lots of resources, including:
    - **CPU** (running time) usage
    - **Memory** (internal) usage
    - **Disk** (external memory) usage
    - **Network** usage
  - All are important but we will mostly talk about the **time complexity** (**CPU** usage).

8

## Algorithm CPU Efficiency

- **Performance vs Complexity?**
  - **Performance:**
    - How much time is used when a program is executed.
    - Please note that this depends on the computer, operating system, compiler, etc.
  - **Complexity:**
    - What happens as the **size** of the problem being solved gets larger?
    - We want to evaluate the "quality" of the algorithms independent of their implementation.

9

## Algorithm Analysis

- **In this module we shall discuss:**
  - How to **estimate** the **running time** required for an **algorithm**.
  - How to **reduce the running time** of an algorithm (from days or years to fractions of a second).
  - The results of careless use of **recursion**.
  - Some examples of **efficient algorithms**.

10

## Algorithm Analysis

- **Very Important!**
  - When we are trying to find the **complexity** of an algorithm, **we are not interested in the exact number of operations** that are necessary to solve the problem.
  - We are interested in **the relation of the number of operations to the problem size**
  - We are usually interested in the **worst case** - the maximum number of operations that are necessary to be performed for a given problem size.

11

## Algorithm Analysis

- Most of the problems we will study have an **input size**.
- Example:
  - For the problem of sorting, the size of the input is the number of elements to be sorted.
- The **CPU** time and **space** (internal **memory**) used by an algorithm will in general be a function of this **input size**.
- For a given input size **n** we often express the time **T** to execute the algorithm as a **function of n**, written as **T(n)**.
- We will always assume **T(n) is a non-negative value function.**

12

## Algorithm Analysis

- **Example:**
  - Consider a simple algorithm to solve the problem of **finding the largest (max) value in an array of n integers**.
  - The algorithm looks at each integer in turn, saving the position of the largest value seen so far.
  - The **total time** to run this algorithm is approximately **cn**, because we must make **n** comparisons, with each comparison costing **c** time.
  - Then for this algorithm **T(n) = cn**



## Algorithm Analysis

- **Example:**
  - An algorithm with running time **T(n) = n**$^2$ requires **1024 × 1024 = 1,048,576** time steps for an input of size **n = 1024**.
  - An algorithm with running time **T(n) = nlogn** requires **1024 × 10 = 10,240** time steps for an input of size **n = 1024**.
  - It's easy to verify that **n**$^2$ **> 10nlogn** when **n > 58**.

You would be **much better** off **changing algorithms** instead of **buying a computer ten times faster**.

15