

G205

# Fundamentals of Computer Engineering

CLASS 1

Stefano Basagni

Fall 2003

M-W, 9:50am-11:30am, 410 EII

# Aims of the Class

- ◆ Basics of data structures and algorithms
- ◆ Resource (e.g., time, space) analysis
- ◆ Algorithm correctness
- ◆ Implementation issues (C++)  
(This is not a C++ class!)

# Algorithms

- ◆ An ALGORITHM is a:
  - Well defined computational procedure
  - INPUT VALUE → OUTPUT VALUES
  - Set of COMPUTATIONAL STEPS to transform the INPUT into the OUTPUT
  - Tool for solving a COMPUTATIONAL PROBLEM

# Computational Problems

- ◆ A *Computational Problem* (CP) is a:
  - General term description of an INPUT/OUTPUT relationship
  - The way from INPUT to OUTPUT (algorithm) is NOT described

# Example: SORTING, 1

◆ As a computational problem:

- INPUT: a sequence of  $n$  numbers

$\langle a_1, a_2, \dots, a_n \rangle$

- OUTPUT: A permutation (reordering)

$\langle a'_1, a'_2, \dots, a'_n \rangle$  on the input sequence such that:

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

# EXAMPLE: Sorting, 2

- ◆ Input sequence:  $\langle 31, 41, 59, 26, 41, 58 \rangle$
- ◆ Output sequence:  $\langle 26, 31, 41, 41, 58, 59 \rangle$
- ◆ The input sequence is called an INSTANCE of the sorting problem
- ◆ One CP  $\rightarrow$  many (sorting) algorithms
  - NEXT QUESTION ...

# The BEST algorithm for a CP

## ◆ Depends on:

- Size of the instance (how many numbers to be sorted?)
- “Format” of the instance (are the numbers sorted already?)
- Restriction on the input values
- Where are the values stored
- The metric of interest (best wrt to what?)

# Algorithm EFFICIENCY, 1

- ◆ How FAST is an algorithm? How much SPACE does it need?
- ◆ *Complexity* of an algorithm, as a function of the SIZE OF THE INPUT
  - Time complexity often more important of space complexity
  - Other complexity metrics (messages)

# Algorithm EFFICIENCY, 2

- ◆ Grossly speaking: An algorithm is EFFICIENT when its time complexity is at most “polynomial”
  - $t(n)$ :  $\log^k n$ ,  $\text{sqrt}(n)$ ,  $n$ ,  $n^k$ ,  $n^k \log^k n$
- ◆ Exponential time complexities are considered “bad”
  - $t(n)$ :  $a^{k(n)}$ ,  $n^n$ ,  $n!$

# Algorithm Correctness

- ◆ An algorithm is said to be **CORRECT** if *for every* input it **HALTS** with the expected, correct output
  - → Termination
  - → Correctness of output
- ◆ A correct algorithm it is said to **SOLVE** a computational problem

# Data Structures

- ◆ Facilitate access and modifications
- ◆ Way to store and organize data, i.e.,  
input, output and intermediate values
- ◆ Impact on algorithm *efficiency*

# From Algorithms to Programs

- ◆ Pseudo-code highlights algorithms properties/requirements
- ◆ One algorithm, many programming languages
- ◆ C++, object orientation + Standard Template library = very close to pseudo-code
- ◆ Executable and understandable

# A Working Example: Sorting $n$ Numbers

- INPUT: a sequence of  $n$  numbers  
 $\langle a_1, a_2, \dots, a_n \rangle$
- OUTPUT: A permutation (reordering)  
 $\langle a'_1, a'_2, \dots, a'_n \rangle$  on the input sequence such  
that:  $a'_1 \leq a'_2 \leq \dots \leq a'_n$
- ◆ Data structure for the input: ARRAY A  
with  $n$  elements
- ◆ Sorting is said to be IN PLACE if  
numbers are rearranged in A

# Insertion Sort, 1

- ◆ Efficient for small numbers of values
- ◆ Sort a hand of playing cards
- ◆ Input is an array  $A[1\dots n]$
- ◆ Sorting in place

# Insertion Sort, 2

Insertion-Sort(A,n)

  for  $j = 2$  to  $n$  do

    key = A[j]

$i = j - 1$

    while (  $i > 0$  ) and (  $A[i] > \text{key}$  ) do

$A[i + 1] = A[i]$

$i = i - 1$

$A[i + 1] = \text{key}$

# Insertion Sort, 3

a) [5,2,4,6,1,3]

b) [2,5,4,6,1,3]

c) [2,4,5,6,1,3]

d) [2,4,5,6,1,3]

e) [1,2,4,5,6,3]

f) [1,2,3,4,5,6]

# Insertion Sort: Correctness, 1

## ◆ Via *loop invariants*

- (\*) At the start of each iteration of the for loop, the sub-array  $A[1 \dots j-1]$  is sorted

## ◆ We have to show three things:

- *Initialization:* (\*) is true before the loop
- *Maintenance:* If (\*) is true before an iteration of the loop, it is true before the next one
- *Termination:* (\*) at the end helps to show the algorithm correctness

# Insertion Sort: Correctness, 2

- ◆ Init:  $j = 2$ ,  $A[1] = 5$  is sorted!
- ◆ Maint: The outer loop seek a position for  $A[j]$  in  $A[1..j-1]$  and insert it in the right position. If  $A[1..j-1]$  is sorted,  $A[1..j]$  is sorted too (cmp. induction)
- ◆ Termin: The loop terminates when  $j=n+1$ . In this case  $A[1..n]$  is sorted and hence the algorithm is correct

# Analysis of Algorithms, 1

- ◆ Analyzing = predicting the resources (here *time*) that the algorithm require
- ◆ Model of computation: one-processor RAM = Random Access Machine
  - Instruction are executed serially
  - No concurrent operations
- ◆ Usual constant time operations: arithmetic, data movements and control

# Analysis of Algorithms, 2

## ◆ RUNNING TIME as a function of the SIZE OF THE INPUT

- Input size:
  - ◆ Number of items in the input (e.g., sorting)
  - ◆ Total number of bits needed to represent the input in the model (e.g., primality)
- Running time: number of primitive operations or “steps” executed

# Insertion Sort: Analysis

Insertion-Sort(A,n)	cost	times
for j = 2 to n do	c1	n
key = A[j]	c2	n-1
i = j-1	c3	n-1
while (i>0) and (A[i]>key) do	c4	(a)
A[i+1] = A[i]	c5	(b)
i = i-1	c6	(c)
A[ i + 1 ] = key	c7	n-1

# Insertion Sort: Running time, 1

- ◆  $t_j$  = number of times the while is executed in the  $j$ -th for loop
- ◆  $(a) = \text{SUM}(j=2, n) t_j$
- ◆  $(b) = (c) = \text{SUM}(j=2, n) (t_j - 1)$
- ◆  $T(n) = c_1 * n + c_2 * (n - 1) + c_3 * (n - 1) + c_4 * (a) + c_5 * (b) + c_6 * (c) + c_7 * (n - 1)$

# Insertion Sort: Running time,2

- ◆ Dependency on the while = dependency on the input
  - BEST CASE: while never executed = array is already sorted ( $t_j=1$ )
    - ◆  $T(n) = Cn+D$ , LINEAR FUNCTION OF  $n$
  - WORST CASE: while always executed = arrays sorted reverse
    - ◆  $T(n) = Cn^2+D$ , QUADRATIC FUNCTION OF  $n$

# Order of Growth

- ◆ Actual cost of single operations can be ignored since it depends on the specific computer, on the language, etc.
- ◆ Another abstraction: Order of growth. We consider the leading term of a formula, with no constants
- ◆ Expressed by the “theta notation”

# Analysis, again

## ◆ Worst case analysis

- Time complexity in the worst case = longest running time for *any* input of size  $n$
- It is an UPPER BOUND on the running time for any input
- INSERTION SORT is  $O(n^2)$ , i.e., quadratic

## ◆ Average case analysis

- A distribution of the input is considered

# Assignments

- ◆ Textbook, till page 27
- ◆ Homework 1: due in class 9/17/2003
- ◆ Updated information on the class web page:

[www.ece.neu.edu/courses/eceg205/2003fa](http://www.ece.neu.edu/courses/eceg205/2003fa)