CSI - 3105 Design & Analysis of Algorithms
Course 1

Jean-Lou De Carufel

Fall 2019

Course Outline

- Website

  http://cglab.ca/~jdecaruf/CSI3105.html
- My office: STE - 5108
- The following textbook is amazing.
  - Sanjoy Dasgupta, Christos H. Papadimitriou and Umesh Vazirani.
    Algorithms.
    McGraw-Hill Education, 2006.
- Course evaluation

Assignment 1:	5%	Sept. 24, 2019
Assignment 2:	5%	Oct. 10, 2019
Assignment 3:	5%	Nov. 19, 2019
Exam 1:	17.5%	**Oct. 8, 2019, 10:00 to 11:20**
Exam 2:	17.5%	**Nov. 12, 2019, 10:00 to 11:20**
Final exam:	50%	TBA
Total:	100%	

- Suggested readings
- Exercises
- Assignments
- Exams
- Some announcement

Chapter 1: Introduction

What does “algorithm” mean?

Al-Khwarizmi (783 - 850)

What does “design & analysis of algorithms” mean?

- Correctness of algorithms.
- Does it terminate?
- Efficient (fast):
  - Estimate the *running time*.
  - Count the number of *steps*.
  - Is it optimal? Can we do better?
- Limits of efficiency (some problems *cannot* be solved efficiently).
- Pseudocode, no programming.

Insertion Sort

Input: An array $A[1..n]$ of n numbers.
Output: An array containing the numbers of A in increasing order.

```
for j = 2 to n do
    key = A[j]
    i = j − 1
    while i > 0 and A[i] > key do
        A[i + 1] = A[i]
        i = i − 1
    end while
    A[i + 1] = key
end for
```

- What is “the best” input for Insertion Sort?
- What it “the worst” input for Insertion Sort?
- How much time does it take to sort n numbers with Insertion Sort ?

Line	Instruction	Time (in ms.)	# of times
1	for $j = 2$ to n do	c_1	n
2	$key = A[j]$	c_2	$n - 1$
3	$i = j - 1$	c_3	$n - 1$
4	while $i > 0$ and $A[i] > key$ do	c_4	$\sum_{j=2}^{n} t_j$
5	$A[i+1] = A[i]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6	$i = i - 1$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	end while	0	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_7	$n - 1$
9	end for	0	$n - 1$

- In the best case, it takes $an + b$ units of time to sort n numbers with Insertion Sort (for some constants a and b).
- In the worst case, it takes $cn^2 + dn + e$ units of time to sort n numbers with Insertion Sort (for some constants c, d and e).

Suppose that a computer can execute $10^9 = 1\,000\,000\,000$ operations per second.

Number of operations	$n = 100$	$n = 1\,000\,000$
n^2	$0,000\,010\,000$ sec.	1000 sec.
$\frac{1}{2}n^2 - \frac{1}{2}n$	$0,000\,005\,000$ sec.	500 sec.
n	$0,000\,000\,100$ sec.	$0,001$ sec.
$\log(n)$	$0,000\,000\,007$ sec.	$0,000\,000\,020$ sec.
2^n	4×10^{13} years	$3 \times 10^{301\,013}$ years

- 4×10^{13} years is larger than the age of the universe.
- What do you think of the constants a, b, c, ... ?

Definition (O-Notation)

Let

$$\begin{aligned} f : \mathbb{N} &\longrightarrow \mathbb{R}^+, \\ g : \mathbb{N} &\longrightarrow \mathbb{R}^+ \end{aligned}$$

be two functions. We say that f *is O of* (or *is big O of*) g if there exist a constant $c \in \mathbb{R}^+$ and a number $k \in \mathbb{N}$ such that $f(n) \leq c\, g(n)$ for all $n \geq k$.
We write

$$f(n) = O(g(n))$$

or

$$f = O(g) \ .$$

Insertion Sort takes $O(n^2)$ time in the worst case.

Definition (Ω-Notation)

Let

$$\begin{aligned} f : \mathbb{N} &\longrightarrow \mathbb{R}^+, \\ g : \mathbb{N} &\longrightarrow \mathbb{R}^+ \end{aligned}$$

be two functions. We say that f *is* Ω *of* (or *is big* Ω *of*) g if there exist a constant $c \in \mathbb{R}^+$ and a number $k \in \mathbb{N}$ such that $f(n) \geq c\, g(n)$ for all $n \geq k$.
We write

$$f(n) = \Omega(g(n))$$

or

$$f = \Omega(g) \ .$$

Insertion Sort takes $\Omega(n^2)$ time in the worst case.

Definition (Θ-Notation)

Let

$$\begin{aligned} f : \mathbb{N} &\longrightarrow \mathbb{R}^+, \\ g : \mathbb{N} &\longrightarrow \mathbb{R}^+ \end{aligned}$$

be two functions. We say that f *is* Θ *of* (or *is big* Θ *of*) g if there exist two constants $c_1 \in \mathbb{R}^+$ and $c_2 \in \mathbb{R}^+$ and a number $k \in \mathbb{N}$ such that $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ for all $n \geq k$.
We write

$$f(n) = \Theta(g(n))$$

or

$$f = \Theta(g) \ .$$

Insertion Sort takes $\Theta(n^2)$ time in the worst case.

An algorithm $\mathcal{A}$ takes $O(T(n))$ time, for a function T, if there exist

- a strictly positive constant c
- and an implementation of $\mathcal{A}$ which takes at most $c\,T(n)$ units of time to execute for any input of size n.

This is possible thanks to the *Principle of Invariance*.

Two different implementations of the same algorithm will not differ in efficiency by more than some multiplicative constant.

Barometer Instruction

A *barometer instruction* is one that is executed at least as often as any other instruction in the algorithm.

There is no harm if some instructions are executed up to a constant number of times more often than the barometer since their contribution is absorbed in the asymptotic notation (O, Ω and/or Θ).

The time of computation of the algorithm is then in the order of the number of executions of the barometer instruction.

Can you identify a barometer instruction in Insertion Sort?

To establish the relation between two functions, we can use the following theorem.

Theorem (Limit Criterion)

Let

$$\begin{aligned} f : \mathbb{N} &\longrightarrow \mathbb{R}^+, \\ g : \mathbb{N} &\longrightarrow \mathbb{R}^+ \end{aligned}$$

be two functions. Let

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)} \ .$$

- *If* $L = 0$, *then* $f = O(g)$ *(and* $f \neq \Theta(g)$*).*
- *If* $L = \infty$, *then* $f = \Omega(g)$ *(and* $f \neq \Theta(g)$*).*
- *If* $L \in \mathbb{R}^+$, *then* $f = \Theta(g)$ *(and* $g = \Theta(f)$*).*
- *If the limit does not exist, then we cannot conclude.*