

CH08-320201

Algorithms and Data Structures

ADS

Lecture 2

Dr. Kinga Lipskoch

Spring 2019

Running Time

- ▶ The running time depends on the input: an already sorted sequence is easier to sort
- ▶ Parameterize running time by the size of the input: short sequences are easier to sort than long ones
- ▶ Generally, we seek upper bounds on the running time: we would like to have a guarantee

Types of Analyses

- ▶ **Worst case** (usually)
 $T(n)$ = maximum time of algorithm on any input of size n
- ▶ **Average case** (sometimes)
 $T(n)$ = expected time of algorithm over all inputs of size n
(Need assumption of statistical distribution of inputs)
- ▶ **Best case** (almost never)
Does not make much sense, e.g., we can start with the solution

Asymptotic Analysis

- ▶ What is Insertion Sort's worst-case time?
 - ▶ It depends on the speed of our computer: relative speed (on the same machine), absolute speed (on different machines)
- ▶ Idea
 - ▶ Ignore machine-dependent constants
 - ▶ Look at growth of $T(n)$ as $n \rightarrow \infty$

Asymptotically Tight Bound: Θ -Notation

For a given asymptotically non-negative function $g(n)$, we define

$$\Theta(g(n)) = \{f(n) \mid \exists \text{ positive constants } c_1, c_2 \text{ and } n_0, \\ \text{such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \forall n \geq n_0\}$$

We often write $f(n) = \Theta(g(n))$

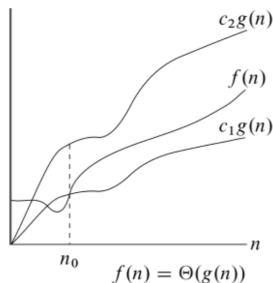
(not an equation, also not an assignment) instead of $f(n) \in \Theta(g(n))$.

The same is meant by both notations.

Example:

$$f(n) = 3n^3 + 90n^2 - 5n + 6046$$

$$3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$$



Example

$$c_1 n^3 \leq 3n^3 + 90n^2 - 5n + 6046 \leq c_2 n^3$$

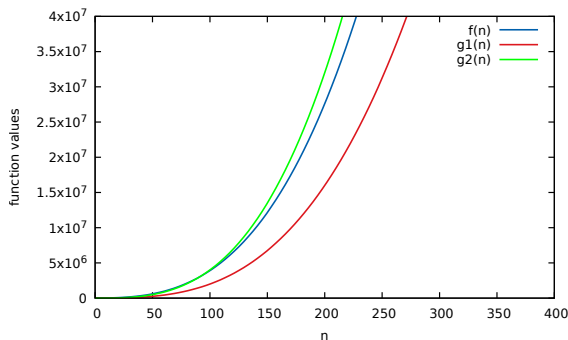
$$c_1 \leq 3 + \frac{90}{n} - \frac{5}{n^2} + \frac{6046}{n^3} \leq c_2$$

$$\text{Try } c_1 = 2; c_2 = 4; n_0 = 100; \Rightarrow f_{div}(n_0) = 3.906546$$

Intuitively:

- ▶ set c_1 to a value smaller than the coefficient of the highest-order term
- ▶ and c_2 to a value that is slight larger

Plotting Functions Using Gnuplot



$$f(n) = 3n^3 + 90n^2 - 5n + 6046$$

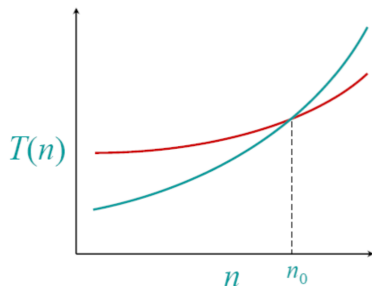
$$g1(n) = 2n^3$$

$$g2(n) = 4n^3$$

Gnuplot script

Asymptotic Performance

- ▶ When n gets large enough, a $\Theta(n^2)$ algorithm always beats a $\Theta(n^3)$ algorithm
- ▶ Informal notion:
 - ▶ throw away lower-order terms
 - ▶ ignore the leading coefficient of the highest-order term



$$3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$$

Asymptotic Analysis

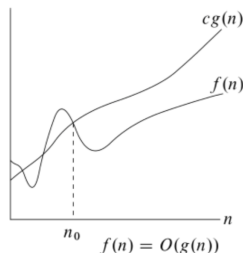
- ▶ We should **not** ignore asymptotically slower algorithms
- ▶ Real-world design situations often call for a careful balancing of engineering objectives
- ▶ Asymptotic analysis is a useful tool to help to structure our thinking

Asymptotically Upper Bound: O -Notation

For a given asymptotically non-negative function $g(n)$, we define $O(g(n)) = \{f(n) | \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$

Example:

We say that $f(n)$ is polynomially bounded if $f(n) = O(n^k)$ for some constant k



Examples

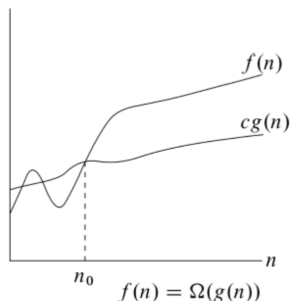
► $f(n) = 3n^3 + 90n^2 - 5n + 6046$
 $\implies f(n) = O(n^3)$

► $f(n) = n$
 $\implies f(n) = O(n^3) ???$
 $\implies f(n) = O(n^2) ???$
 $\implies f(n) = O(n)$ also true

Asymptotically Lower Bound: Ω -Notation

For a given asymptotically non-negative function $g(n)$, we define

$$\Omega(g(n)) = \{f(n) \mid \exists \text{ positive constants } c \text{ and } n_0, \\ \text{such that } 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$



For tight bounds, we get
 $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Non-tight Upper Bound: o -Notation

For a given asymptotically non-negative function $g(n)$, we define $o(g(n)) = \{f(n) \mid \text{for any constant } c > 0, \exists n_0 > 0, \text{ such that } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$

$$f(n) = o(g(n)) \text{ implies } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Examples

► $2n = o(n^2)$

► $2n^2 \neq o(n^2) ???$

► $n^b = o(a^n)$ for $a > 1$

Non-tight Lower Bound: ω -Notation

For a given asymptotically non-negative function $g(n)$, we define $f(n) \in \omega(g(n))$ iff $g(n) \in o(f(n))$

$$f(n) = \omega(g(n)) \text{ implies } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

Asymptotic Analysis of Insertion Sort (1)

INSERTION-SORT(A, n)	<i>cost</i>	<i>times</i>
for $j = 2$ to n	c_1	n
$key = A[j]$	c_2	$n - 1$
// Insert $A[j]$ into the sorted sequence $A[1..j - 1]$.	0	$n - 1$
$i = j - 1$	c_4	$n - 1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^n t_j$
$A[i + 1] = A[i]$	c_6	$\sum_{j=2}^n (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^n (t_j - 1)$
$A[i + 1] = key$	c_8	$n - 1$

- ▶ c_k is the number of steps a computer needs to perform instruction k once (e.g., a Random Access Machine)
- ▶ t_j is the number of times the while loop is executed in the for iteration j

Asymptotic Analysis of Insertion Sort (2)

- ▶ Best case:
Input series is ordered.
- ▶ Then, $t_j = 1$.
- ▶ $T(n) = \Theta(n)$

Asymptotic Analysis of Insertion Sort (3)

- ▶ **Worst case:**

Input series was ordered in reverse.

- ▶ Then, $t_j = j$.

- ▶ With the arithmetic series

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

the worst-case asymptotic complexity is

$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2)$$

Asymptotic Analysis of Insertion Sort (4)

- ▶ **Average case:**

All permutations are equally likely.

- ▶ Then, t_j is expected to be $\frac{j}{2}$ on average.

- ▶ Hence, the average-case asymptotic complexity is

$$T(n) = \sum_{j=2}^n \Theta\left(\frac{j}{2}\right) == \Theta(n^2)$$

Asymptotic Analysis of Insertion Sort (5)

- ▶ Is Insertion Sort fast?
- ▶ For small n , it is moderately fast.
- ▶ For large n , it is slow.

Summary: Asymptotic Analysis

- ▶ O -notation – Asymptotically upper bound
 - ▶ Ω -notation – Asymptotically lower bound
 - ▶ Θ -notation – Asymptotically tight bound
 - ▶ o -notation – Non-tight upper bound
 - ▶ ω -notation – Non-tight lower bound
-
- ▶ $f(n) = O(g(n))$ is like $a \leq b$,
 - ▶ $f(n) = \Omega(g(n))$ is like $a \geq b$,
 - ▶ $f(n) = \Theta(g(n))$ is like $a = b$,
 - ▶ $f(n) = o(g(n))$ is like $a < b$,
 - ▶ $f(n) = \omega(g(n))$ is like $a > b$.