Analysis of Algorithms
Master MOSIG - Algorithms and Program Design

Marc Tchiboukdjian - Denis Trystram

25-09-2009

Course

Objectives: Analyze the cost of an algorithm

To Remember

Correctness of an algorithm. Use a *loop invariant* and prove

- *Initialization:* it is true prior to the first iteration of the loop.
- *Maintenance:* if it is true before an iteration, it remains true before the next iteration.
- *Termination:* when the loop terminates, it gives a useful property to prove correctness.

Cost of an algorithm. Resources that the algorithm requires in function of its input i. Most often the **number of primitive operations** or steps executed $T(i)$ or the **memory** used $S(i)$.

Best-case. Worst-case. Average-case. If I is the set of all possible inputs and $|i|$ is the size of input i then

$$T_{best}(n) = \min_{i \in I, |i|=n} T(i)$$
$$T_{worst}(n) = \max_{i \in I, |i|=n} T(i)$$
$$T_{average}(n) = \sum_{i \in I, |i|=n} T(i) \Pr\{\text{input } i \text{ occured}\}$$

Asymptotic notation. Drop lower-order terms. Ignore the constant coefficient in the leading term.

$$f = \mathrm{O}(g) \Leftrightarrow \exists C > 0\ \exists n_0 > 0\ \forall n > n_0\ f(n) \leq Cg(n)$$
$$f = \Omega(g) \Leftrightarrow g = \mathrm{O}(f)$$
$$f = \Theta(g) \Leftrightarrow f = \mathrm{O}(g) \text{ and } f = \Omega(g)$$
$$f = \mathrm{o}(g) \Leftrightarrow \forall C > 0\ \exists n_0 > 0\ \forall n > n_0\ f(n) \leq Cg(n)$$
$$f = \omega(g) \Leftrightarrow g = \mathrm{o}(f)$$

Useful properties about logarithms

$$\begin{aligned}
\log_b b^a &= a \\
\log_c a.b &= \log_c a + \log_c b \\
\log_b a^n &= n \log_b a \\
\log_b a &= \frac{\log_c a}{\log_c b} \\
\log_b a &= \mathrm{O}(\log a)
\end{aligned}$$

Example

Asymptotic notation

$$\begin{aligned}
2n^2 + 3n + \log n &= \Theta(n^2) \\
&= \mathrm{O}(n^3) \\
&= \Omega(n \log n) \\
&= \mathrm{o}(n^3) \\
&= \omega(n)
\end{aligned}$$

O	$\leq$
Ω	$\geq$
o	$<$
ω	$>$
Θ	$=$

Analysis of insertion sort

Algorithm 1: Insertion sort

Input: An array A of n integers
Result: The array A is sorted

```
1 for j = 2 to n do
2 |   key ← A[j]
3 |   i ← j − 1
4 |   while i > 0 and A[i] > key do
5 |   |   A[i + 1] ← A[i]
6 |   |   i ← i − 1
7 |   end
8 |   A[i + 1] ← key
9 end
```

Best-case: the array is already sorted.

- Always find that $A[i] \leq key$ upon the first time the while loop test is run (when $i = j - 1$).
- The body of the for loop (lines 2-8) has cost O(1).

- The for loop is executed $\mathrm{O}(n)$ times.
- $T_{best}(n) = \mathrm{O}(n)$.

Worst-case: the array is sorted in the reverse order.

- Always find that $A[i] > key$ in while loop test.
- Have to compare key with all elements to the left of the jth position: compare with $j - 1$ elements.
- The body of the for loop has cost $\mathrm{O}(j)$.
- $T_{worst}(n) = \sum_{j=2}^{j=n} \mathrm{O}(j) = \mathrm{O}(n^2)$.

Average-case: assume each input is equiprobable.

- Equiprobable inputs imply for each element, rank among elements so far is equiprobable.
- When inserting element in jth position, the expected number of times the while loop is executed is $\sum_{k=1}^{k=j} k/j = \mathrm{O}(j)$.
- $T_{average}(n) = \sum_{j=2}^{j=n} \mathrm{O}(j) = \mathrm{O}(n^2)$.

Exercises

Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	$\sqrt{n}$	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					

Linear search

Consider the searching problem:
Input: An array A of n numbers and a value v.
Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

Write pseudocode for linear search, which scans through the sequence, looking for v. How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in Θ notation?

Selection sort

Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n - 1$ elements of A.

Write pseudocode for this algorithm, which is known as selection sort. Why does it need to run for only the first n - 1 elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ notation.