AI Lab-Class
Algorithm Analysis

Maximilian Schöbel

Hochschule Bonn-Rhein-Sieg

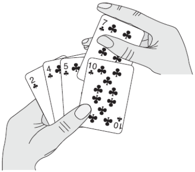
October 17, 2018

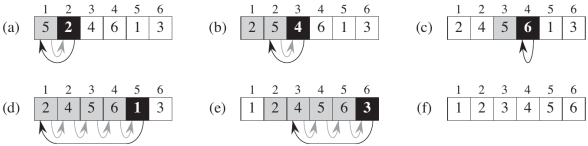
Asymptotic Algorithm Analysis

"Duuude, yesterday I implemented an algorithm that sorts an array in only 10 seconds!!!" Nice, **but:**

- Performance and runtime of an algorithm depend on the **input** and available **hardware**.
- Not only the input size matters, but also the properties of the input, e.g. if it is pre-sorted or somehow ordered in advance.

Motivation: Insertion Sort's running time






```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Motivation: Insertion Sort's running time

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

- t_j: number of times the while-condition is tested in iteration j
- Best case (array pre-sorted): $t_j = 1$
- Worst case (array sorted in descending order): $t_j = j - 1$

Motivation: Insertion Sort's running time

General case:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
&\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \,.
\end{aligned}
$$

Best case ($t_j = 1$):

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \,.
\end{aligned}
$$

Worst case ($t_j = j - 1$):

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8) \,.
\end{aligned}
$$

Concentrate on the worst case!

- Worst-case running time gives an upper bound for any input.
  A guarantee, that the algorithm will never take longer!
- The worst case appears fairly often (database lookups).
- The average case is often roughly as bad as the worst case.

Motivation: Asymptotic algorithm analysis

- Precise analysis of an algorithm in a computational model is tedious. So far, we haven't even touched recursive algorithms!
- Needed: A metric that is simple to write and manipulate, shows the most important resource requirements while suppressing the tedious details.
- To the rescue: Asymptotic notation!

O-Notation

- Let $g : \mathbb{N} \to \mathbb{R}_+$ be an arbitrary function.
- The set of functions $f : \mathbb{N} \to \mathbb{R}_+$, which do not grow faster than the function g after a specific point n_0, is denoted as $O(g(n))$.
- More specifically:
  $O(g(n)) := \{f(n) \mid \exists\ c \in \mathbb{R} \text{ and } \exists\ n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0\}$

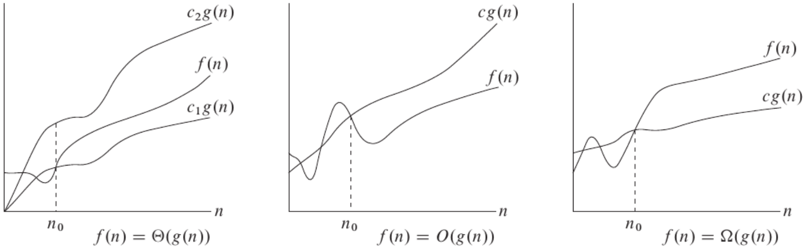


Figure 1: O, Θ and Ω notation