Principles of Algorithmic Techniques
T-79.4202

Pekka Orponen
Department of Computer Science
Aalto University

Autumn 2015
In algorithm design, it is important to understand how the resource requirements of an algorithm depend on the input characteristics.

A standard measure is an algorithm’s worst-case runtime, or “time complexity”.

The dependence of this on input size can typically be determined \textit{a priori}, by just analysing the algorithm description. (This can, however, sometimes be a challenging mathematical problem.)
2.1 Analysis of algorithms: basic notions

- Generally speaking, an algorithm $A$ computes a mapping from a potentially infinite set of inputs (or instances) to a corresponding set of outputs (or solutions).
- Denote by $T(x)$ the number of elementary operations that $A$ performs on input $x$ and by $|x|$ the size of an input instance $x$.
- Denote by $T(n)$ also the worst-case time that algorithm $A$ requires on inputs of size $n$, i.e.,

\[ T(n) = \max \{ T(x) : |x| = n \}. \]
2.2 Analysis of iterative algorithms

To illustrate the basic worst-case analysis of iterative algorithms, let us consider yet another well-known (but bad) sorting method.

Algorithm 1: The insertion sort algorithm

1 function INSERTSORT (A[1 . . . n])
2 for i ← 2 to n do
3     a ← A[i]; j ← i − 1
4     while j > 0 and a < A[j] do
5         A[j + 1] ← A[j]; j ← j − 1
6     end
7     A[j + 1] ← a
8 end
Analysis of insertion sort

Denote: $T_{k-l}$ = the complexity of a single execution of lines $k$ thru $l$. Then:

\[
T_5(n, i, j) \leq c_1
\]
\[
T_{4-6}(n, i) \leq c_2 + (i - 1)c_1
\]
\[
T_{3-7}(n, i) \leq c_3 + c_2 + (i - 1)c_1
\]
\[
T_{2-8}(n) \leq c_4 + \sum_{i=2}^{n} (c_3 + c_2 + (i - 1)c_1)
\]
\[
= c_4 + (n - 1)(c_3 + c_2) + c_1 \sum_{i=2}^{n} (i - 1)
\]
\[
\leq \text{const} \cdot n + c_1 \cdot \frac{1}{2} n(n - 1)
\]

Thus $T(n) = T_{2-8}(n) = O(n^2)$. 
Basic analysis rules

Denote $T[P]$ = the time complexity of an algorithm segment $P$.

- $T[x \leftarrow e] = \text{constant}$, $T[\text{read } x] = \text{constant}$, $T[\text{write } x] = \text{constant}$.
- $T[S_1; S_2; \ldots; S_k] = T[S_1] + \cdots + T[S_k] = O(\max\{T[S_1], \ldots, T[S_k]\})$
- $T[\text{if } P \text{ then } S_1 \text{ else } S_2] = \begin{cases} T[P] + T[S_1] & \text{if } P = \text{true} \\ T[P] + T[S_2] & \text{if } P = \text{false} \end{cases}$
- $T[\text{while } P \text{ do } S] = T[P] + (\text{number of times } P = \text{true}) \cdot (T[S] + T[P])$

In analysing nested loops, proceed from innermost out. Control variables of outer loops enter as parameters in the analysis of inner loops.
Operation costs in programming languages

- If one is using a modern high-level programming language (such as Python), one needs to be aware of the runtime costs of basic operations.
- If an operation addresses a structured object, the actual cost may depend highly on the language implementation.
- E.g. if \( A \) is a Python list of length \( n \), then the cost of obtaining a \( k \)-element “slice” such as \( A[0:k] \) is \( \mathcal{O}(k) \).
- In fact, even the cost of *inserting a single element* at or near the beginning of the list, e.g. operation \( A.insert(0,x) \) has cost \( \mathcal{O}(n) \)! This is because Python lists are implemented internally as simple linear arrays.
- The costs of several basic Python operations on structured data are listed at [https://wiki.python.org/moin/TimeComplexity](https://wiki.python.org/moin/TimeComplexity).
2.3 Analysis of recursive algorithms

Let us review the analysis of the merge sort algorithm.

Let \( \text{MERGE}(A', A'', A) \) be a subroutine which takes two sorted arrays \( A'[1..m] \) and \( A''[1..(n – m)] \) and merges them, element by element, into a single sorted sequence in array \( A[1..n] \) in \( \mathcal{O}(n) \) steps.

**Algorithm 2: The merge sort algorithm**

1. **function** \( \text{MERGE} \text{SORT} (A[1 \ldots n]) \)
2. if \( n = 1 \) then return else
3. Introduce auxiliary arrays \( A'[1..\lfloor n/2 \rfloor], A''[1..\lceil n/2 \rceil] \)
4. \( A' \leftarrow A[1..\lfloor n/2 \rfloor] \)
5. \( A'' \leftarrow A[\lceil n/2 \rceil + 1..n] \)
6. \( \text{MERGE} \text{SORT}(A') \)
7. \( \text{MERGE} \text{SORT}(A'') \)
8. \( \text{MERGE}(A', A'', A) \)
9. end
Assume the number of elements $n$ is a power of 2. Then the runtime $T(n)$ of MERGESORT can be described by a recurrence relation:

$$
\begin{cases}
T(1) \leq c_1 \\
T(n) \leq 2T\left(\frac{n}{2}\right) + c_2 n, \quad n = 2^k, \ k = 0, 1, 2, \ldots
\end{cases}
$$

A straightforward approach to solving such recurrences is by “unwinding” them:

\[
T(n) \leq 2T\left(\frac{n}{2}\right) + c_2 n
\leq 2\left(2T\left(\frac{n}{4}\right) + c_2 \frac{n}{2}\right) + c_2 n
\leq 4T\left(\frac{n}{4}\right) + 2c_2 n
\leq 2^kT\left(\frac{n}{2^k}\right) + k \cdot c_2 n
\leq 2^{\log_2 n}T(1) + \log_2 n \cdot c_2 n
\leq c_1 n + c_2 n \log_2 n = O(n \log_2 n).
\]
2.4 Order of growth of functions

- Also known as the “big-\(O\)” notation.
- In the analysis of algorithms, attention is primarily paid to the growth rate of the dominant term of \(T(n)\), because:
  1. For large input sizes \(n\), the other effects of \(T(n)\) are vanishingly small; and
  2. Different machine implementations of an algorithm induce different constant factors and lower-order terms in \(T(n)\) in any case.
  3. The big-\(O\) notation is a convenient estimation tool.
- We have seen several examples of this usage already, so let’s make the notation precise.
Definitions and notations

- Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$.  
- Define the order (of growth class) of $f$ as:

$$\mathcal{O}(f) = \{t : \mathbb{N} \rightarrow \mathbb{R}^+|\exists c, n_0 \geq 0 : n \geq n_0 \Rightarrow t(n) \leq c \cdot f(n)\}.$$  
- Thus: $t \in \mathcal{O}(f)$, if $t(n)$ is for ”all large enough” $n$ at most a constant factor bigger than $f(n)$.  
- One usually uses the somewhat imprecise, but very convenient notation $t = \mathcal{O}(f)$ instead of the formal $t \in \mathcal{O}(f)$.  
- Also, one denotes functions by means of a “generic argument” $n$. Thus one writes, e.g.,

$$2n^2 + 3n = \mathcal{O}(n^2)$$

as a shorthand for

”the function $f$, for which $f(n) = 2n^2 + 3n$, is in the class $\mathcal{O}(g)$, where $g(n) = n^2$”. 
Examples (1/2)

1. $n^2 = \mathcal{O}(n^3)$ — i.e., to be precise, “$n^2 \in \mathcal{O}(n^3)$”
2. $5n^3 + 3n^2 = \mathcal{O}(n^3)$
3. $\log_a n = \mathcal{O}(\log_b n)$ for all $a, b > 1$
4. $\log_2(n!) = \sum_{i=1}^{n} \log_2 i = \mathcal{O}(n \log_2 n)$
5. $\sum_{i=0}^{n} \frac{1}{2^i} = \mathcal{O}(1) \leftarrow \text{constant}$
6. If $r < s$, then $n^r = \mathcal{O}(n^s)$, $n^s \neq \mathcal{O}(n^r)$
7. If $r < s$, then $r^n = \mathcal{O}(s^n)$, $s^n \neq \mathcal{O}(r^n)$
8. For any $r, s > 1$, $n^s = \mathcal{O}(r^n)$, $r^n \neq \mathcal{O}(n^s)$
9. For any $r, s > 0$, $(\log_2 n)^s = \mathcal{O}(n^r)$, $n^r \neq \mathcal{O}((\log_2 n)^s)$
Examples (2/2)

Figure: Some typical function growth rates.
Properties of order classes

1. \( f(n) \in \mathcal{O}(g(n)), g(n) \in \mathcal{O}(h(n)) \Rightarrow f(n) \in \mathcal{O}(h(n)) \).
2. \( \mathcal{O}(cf(n)) = \mathcal{O}(f(n)) \), for any constant \( c > 0 \).
3. \( \mathcal{O}(f(n) + g(n)) = \mathcal{O}(\max(f(n), g(n))) \).
4. \( f(n) \in \mathcal{O}(g(n)) \Rightarrow \mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n)) \).

Example:

\[
 n^3 + 3n^2 + n + 8 \in \mathcal{O}\left( \frac{n^3}{g(n)} + \frac{3n^2}{f(n)} + n + 8 \right) \subseteq \mathcal{O}(n^3)
\]

Or, more standardly, “\( n^3 + 3n^2 + n + 8 = \mathcal{O}(n^3) \)”.
Comparing orders via limits

**Theorem 1** Let \( f, g : \mathbb{N} \rightarrow \mathbb{R}^+ \). Then:

(i) If

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0,
\]

then \( f(n) = \mathcal{O}(g(n)) \), \( g(n) = \mathcal{O}(f(n)) \).

(ii) If

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,
\]

then \( f(n) = \mathcal{O}(g(n)) \), \( g(n) \neq \mathcal{O}(f(n)) \).

(iii) If

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty,
\]

then \( g(n) = \mathcal{O}(f(n)) \), \( f(n) \neq \mathcal{O}(g(n)) \).
L'Hôpital’s rule

**Theorem 2** Let \( f, g : \mathbb{R} \to \mathbb{R} \). Assume that

\[
\lim_{x \to \infty} f(x) = \lim_{x \to \infty} g(x) = 0 \text{ or } \infty,
\]

and moreover that \( f'(x) \) and \( g'(x) \) exist and \( g'(x) \neq 0 \) for all large enough \( x \). Then

\[
\lim_{x \to \infty} \frac{f(x)}{g(x)} = \lim_{x \to \infty} \frac{f'(x)}{g'(x)},
\]

if the latter limit exists.

**Example.** Let us verify that \( \ln n = \mathcal{O}(\sqrt{n}) \), but \( \sqrt{n} \neq \mathcal{O}(\ln n) \).

\[
D(\ln x) = x^{-1}, \quad D(\sqrt{x}) = \frac{1}{2}x^{-\frac{1}{2}} \neq 0.
\]

\[
\therefore \lim_{n \to \infty} \frac{\ln n}{\sqrt{n}} = \lim_{n \to \infty} \frac{n^{-1}}{\frac{1}{2}n^{-\frac{1}{2}}} = \lim_{n \to \infty} 2n^{-\frac{1}{2}} = 0.
\]

The claim follows by Theorem 1(ii).
Other order-of-growth notions (1/2)

Let \( f : \mathbb{N} \rightarrow \mathbb{R}^+ \). Define the function classes

\[
\Theta(f) = \{ t : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c > 0 : \frac{1}{c} f(n) \leq t(n) \leq cf(n) \text{ for all lar. en. } n \} \\
o(f) = \{ t : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \forall c > 0 : t(n) < cf(n) \text{ for all large enough } n \} \\
\Omega(f) = \{ t : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c > 0 : t(n) \geq cf(n) \text{ for infinitely many } n \}
\]

Some properties:

1. \( f \in \Theta(g) \iff f \in O(g) \text{ and } g \in O(f) \)
2. \( f \in o(g) \iff f \in O(g) \text{ and } g \notin O(f) \)
3. \( f \in \Omega(g) \iff f \notin o(g) \)
Other order-of-growth notions (2/2)

One reads:

- $f \in \Theta(g)$: $f$ is of the same order as $g$.
- $f \in o(g)$: $f$ is of lower order than $g$.
- $f \in \Omega(g)$: $f$ is of at least the same order as $g$.

**Note.** Often in the literature (incl. in the Dasgupta et al. textbook) one sees for the $\Omega$ notation a different definition: $f \in \Omega(g) \iff g \in O(f)$. The difference lies in whether $f$ is required to be bigger than $g$ for “infinitely many $n$” or “all large enough $n$”.
Combining order-of-growth classes (1/2)

Let $X$, $Y$ be function classes (e.g. $X = \mathcal{O}(f)$, $Y = \mathcal{O}(g)$), and let $\circ$ be some arithmetic operation ($+$, $-$, $\ast$, $\ldots$). Define:

$$X \circ Y = \{ t(n) \mid t(n) = f(n) \circ g(n), f(n) \in X, g(n) \in Y \}.$$

Examples:

1. $\mathcal{O}(n^2) + \mathcal{O}(n^3) = \mathcal{O}(n^3)$
2. $n + \mathcal{O}(\frac{1}{n}) \triangleq \{ n \} + \mathcal{O}(\frac{1}{n})$
3. $n \cdot \mathcal{O}(\frac{1}{n}) = \mathcal{O}(n \cdot \frac{1}{n}) = \mathcal{O}(1)$
Combining order-of-growth classes (2/2)

Define also:

\[ \mathcal{O}(X) = \bigcup_{f \in X} \mathcal{O}(f), \]

and correspondingly for the other order-of-growth notions.

**Examples:**

1. \( \mathcal{O}(\mathcal{O}(n^2)) = \mathcal{O}(n^2) \)
2. \( \mathcal{O}(n \cdot \mathcal{O}(n^2)) = \mathcal{O}(\mathcal{O}(n^3)) = \mathcal{O}(n^3) \)
Computing with orders-of-growth (1/2)

1. \( f(n) \in O(f(n)) \)
2. \( c \cdot O(f(n)) = O(f(n)) \) for any constant \( c > 0 \).
3. \( O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(\max(f(n), g(n))) \)
4. \( O(O(f(n))) = O(f(n)) \)
5. \( f(n) \cdot O(g(n)) = O(f(n) \cdot g(n)) \)
6. \( O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n)) \)
Example: What is the dominating behaviour of $n(\sqrt{n} - 1)$ as $n \to \infty$?

\[
\sqrt{n} = e^{\frac{\ln n}{n}} = 1 + \frac{\ln n}{n} + O\left(\left(\frac{\ln n}{n}\right)^2\right)
\]

\[
\therefore n(\sqrt{n} - 1) = n \left(\frac{\ln n}{n} + O\left(\left(\frac{\ln n}{n}\right)^2\right)\right)
\]

\[
= \ln n + n \cdot O\left(\left(\frac{\ln n}{n}\right)^2\right)
\]

\[
= \ln n + O\left(\frac{\ln^2 n}{n}\right)
\]

Consequently: The value of $n(\sqrt{n} - 1)$ grows approximately as $\ln n$. The order of the error in this estimate is $O\left(\frac{\ln^2 n}{n}\right)$, which goes to zero as $n \to \infty$. 