Lecture 2: $O$-notation (Why Constants Matter Less)

COMS10007 - Algorithms

Dr. Christian Konrad

29.01.2019
Runtime of Algorithms

Runtime of an Algorithm

- Function that maps the input length $n$ to the number of simple/unit/elementary operations
- The number of array accesses in Peak Finding represents the number of unit operations very well

Which runtime is better?

- $4(n - 1)$ (simple peak finding algorithm)
- $5 \log n$ (fast peak finding algorithm)
- $0.1n^2$
- $n \log(0.5n)$
- $0.01 \cdot 2^n$

Answer:
It depends... But there is a favourite
\[0.1 n^2 \leq 0.01 \cdot 2^n \leq 5 \log n \leq n \log(n/2) \leq 4(n - 1)\]

\[(n = 10)\]
5 \log n \leq 0.1n^2 \leq n \log(n/2) \leq 4(n - 1) \leq 0.01 \cdot 2^n 
(n = 15)
Runtime Comparisons

\[ 5 \log n \leq n \log(n/2) \leq 0.1n^2 \leq 4(n - 1) \]

\( n = 30 \)
5 \log n \leq n \log(n/2) \leq 4(n - 1) \leq 0.1n^2

(n = 50)
\[ 5 \log n \leq 4(n - 1) \leq n \log(n/2) \leq 0.1n^2 \]

\[ (n = 200) \]
Order Functions Disregarding Constants

**Aim:** We would like to sort algorithms according to their runtime

Is algorithm $A$ faster than algorithm $B$?

**Asymptotic Complexity**
- For large enough $n$, constants seem to matter less
- For small values of $n$, most algorithms are fast anyway (not always true!)

**Solution:** Consider asymptotic behavior of functions

An increasing function $f : \mathbb{N} \to \mathbb{N}$ grows *asymptotically at least as fast as* an increasing function $g : \mathbb{N} \to \mathbb{N}$ if there exists an $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ it holds:

$$f(n) \geq g(n) .$$
Example: $f$ grows at least as fast as $g$
Example: \( f(n) = 2n^3, \ g(n) = \frac{1}{2} \cdot 2^n \)

Then \( g(n) \) grows asymptotically at least as fast as \( f(n) \) since for every \( n \geq 16 \) we have \( g(n) \geq f(n) \)

Proof: Find values of \( n \) for which the following holds:

\[
\frac{1}{2} \cdot 2^n \geq 2n^3 \\
2^{n-1} \geq 2^{3 \log n + 1} \quad \text{(using } n = 2^{\log n} \text{)} \\
n - 1 \geq 3 \log n + 1 \\
n \geq 3 \log n + 2
\]

This holds for every \( n \geq 16 \) (which follows from the racetrack principle). Thus, we chose any \( n_0 \geq 16 \).
**Racetrack Principle:** Let $f, g$ be functions, $k$ an integer and suppose that the following holds:

1. $f(k) \geq g(k)$ and
2. $f'(n) \geq g'(n)$ for every $n \geq k$.

Then for every $n \geq k$, it holds that $f(n) \geq g(n)$.

**Example:** $n \geq 3 \log n + 2$ holds for every $n \geq 16$

- $n \geq 3 \log n + 2$ holds for $n = 16$
- We have: $(n)' = 1$ and $(3 \log n + 2)' = \frac{3}{n \ln 2} < \frac{1}{2}$ for every $n \geq 16$. The result follows.
If \( \leq \) means grows asymptotically at least as fast as then we get:

\[
5 \log n \leq 4(n - 1) \leq n \log(n/2) \leq 0.1n^2 \leq 0.01 \cdot 2^n
\]

Observe:

“polynomial of logarithms” \( \leq \) “polynomial” \( \leq \) “exponential”
**Big O Notation**

**Definition:** $O$-notation ("Big O")

Let $g : \mathbb{N} \to \mathbb{N}$ be a function. Then $O(g(n))$ is the set of functions:

$$O(g(n)) = \{ f(n) : \text{There exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$$

**Meaning:** $f(n) \in O(g(n))$ : "$g$ grows asymptotically at least as fast as $f$ up to constants"
Example: $f(n) = \frac{1}{2} n^2 - 10n$ and $g(n) = 2n^2$
**Example:** \( f(n) = \frac{1}{2} n^2 - 10n \) and \( g(n) = 2n^2 \)

Then: \( g(n) \in O(f(n)) \), since \( 6f(n) \geq g(n) \), for every \( n \geq n_0 = 60 \)
Recall:

\[ O(g(n)) = \{ f(n) : \text{There exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \]

Exercises:

- \( 100n \in O(n) \) Yes, chose \( c = 100, n_0 = 1 \)

- \( 0.5n \in O(n/\log n) \) No: Suppose that such constants \( c \) and \( n_0 \) exist. Then, for every \( n \geq n_0 \):

  \[
  0.5n \leq \frac{cn}{\log n} \\
  \log n \leq 2c \\
  n \leq 2^{2c}, \text{ a contradiction,}
  \]

  since this does not hold for every \( n > 2^{2c} \).
Properties

Recipe

- To prove \( f \in O(g) \): We need to find constants \( c, n_0 \) as in the statement of the definition
- To prove \( f \notin O(g) \): We assume that constants \( c, n_0 \) exist and derive a contradiction

Constants 100 \( \in \ O(1) \) yes, every constant is in \( O(1) \)

Lemma (Sum of Two Functions)

Suppose that \( f, g \in O(h) \). Then: \( f + g \in O(h) \).

Proof. Let \( c, n_0 \) be such that \( f(n) \leq ch(n) \), for every \( n \geq n_0 \). Let \( c', n_0' \) be such that \( g(n) \leq c'h(n) \), for every \( n \geq n_0' \).

Let \( C = c + c' \) and let \( N_0 = \max\{n_0, n_0'\} \). Then:

\[
f(n) + g(n) \leq ch(n) + c'h(n) = Ch(n) \quad \text{for every} \quad n \geq N_0.
\]

\( \square \)
Further Properties

**Lemma (Polynomials)**

Let \( f(n) = c_0 + c_1 n + c_2 n^2 + c_3 n^3 + \cdots + c_k n^k \), for some integer \( k \) that is independent of \( n \). Then: \( f(n) \in O(n^k) \).

**Proof:** Apply statement on last slide \( O(1) \) times (\( k \) times)

**Attention:** Wrong proof of \( n^2 \in O(n) \): (this is clear wrong)

\[
\begin{align*}
n^2 &= n + n + \underbrace{n + \cdots n}_{n-2 \text{ times}} = O(n) + O(n) + \underbrace{n + \cdots n}_{n-2 \text{ times}} \\
&= O(n) + \underbrace{n + \cdots n}_{n-2 \text{ times}} = O(n) + O(n) + \underbrace{n + \cdots n}_{n-3 \text{ times}} = \\
&= O(n) + \underbrace{n + \cdots n}_{n-3 \text{ times}} = \cdots = O(n) .
\end{align*}
\]

Application of statement on last slide \( n \) times!
Tool for the Analysis of Algorithms

- We will express the runtime of algorithms using $O$-notation
- This allows us to compare the runtimes of algorithms
- **Important:** Find the slowest growing function $f$ such that our runtime is in $O(f)$ (most algorithms have a runtime of $O(2^n)$)

Important Properties for the Analysis of Algorithms

- Composition of instructions:
  \[
  f \in O(h_1), g \in O(h_2) \text{ then } f + g \in O(h_1 + h_2)
  \]

- Loops: (repetition of instructions)
  \[
  f \in O(h_1), g \in O(h_2) \text{ then } f \cdot g \in O(h_1 \cdot h_2)
  \]
Hierachy

**Rough incomplete Hierachy**

- Constant time: $O(1)$ (individual operations)
- Sub-logarithmic time: e.g., $O(\log \log n)$
- Logarithmic time: $O(\log n)$ (**FAST-PEAK-FINDING**)
- Poly-logarithmic time: e.g., $O(\log^2 n), O(\log^{10} n), \ldots$
- Linear time: $O(n)$ (e.g., time to read the input)
- Quadratic time: $O(n^2)$ (potentially slow on big inputs)
- Polynomial time: $O(n^c)$ (used to be considered efficient)
- Exponential time: $O(2^n)$ (works only on very small inputs)
- Super-exponential time: e.g. $O(2^{2^n})$ (big trouble...