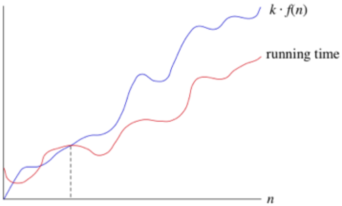
Algorithms and Data Structures

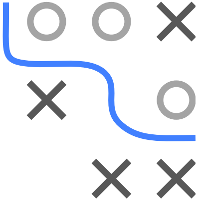
Big O
Misconceptions of Big O, further Landau Symbols & Discussion



Learning goals

- Misconceptions of Big O
- Alternative notations
- Complexity vs. empirical runtime

MISCONCEPTIONS OF BIG O



- **Misconception 1**: $f = \mathcal{O}(g)$: The sign of equality means equality
  - Left: Function
  - Right: Function class $\rightarrow$ equality makes no sense
  - Formally correct: $f \in \mathcal{O}(g)$
- **Misconception 2**: Big O means that functions "have approximately the same" runtime behaviour
  - $f \in \mathcal{O}(1)$ implies by definition also $f \in \mathcal{O}(n)$
  - $f \in \mathcal{O}(g)$ only means that f does not grow faster than g, but not that f grows as fast as g

MISCONCEPTIONS OF BIG O / 2

- **Misconception 3**: Big O describes the runtime of an algorithm
  - Big O describes how well an algorithm scales
  - Big O is not an absolute measure of runtime - an algorithm can have a shorter runtime for a small instance, but scale much worse
- **Misconception 4**: Big O is always the worst case
  - The notation is often used to describe the worst case
  - However Big O does not imply the worst case
  - Also best case and average case can be considered

ALTERNATIVE NOTATIONS

In addition to Big O notation another Landau symbol is used in mathematics: The little o.
Informally $f(x) = o(g(x))$ means that f grows much slower than g.

Formal definition:

$$f(x) \in o(g(x))$$

if and only if
for each $M > 0$ there exists x_0 such that

$$|f(x)| < M \cdot |g(x)| \quad \text{for all } x > x_0.$$

ALTERNATIVE NOTATIONS / 2

Further we define for $a \in \mathbb{R}$

$$f(x) \in o(g(x)) \quad \text{for } x \to a$$

only if for every $M > 0$ there is a $d \in \mathbb{R}$ such that for all x we have $|x - a| < d$

$$|f(x)| < M \cdot |g(x)|$$

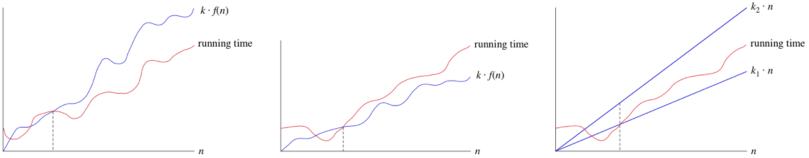
For $g(x) \neq 0$, it is equivalent to

$$\lim_{x \to a} \left| \frac{f(x)}{g(x)} \right| = 0$$

ALTERNATIVE NOTATIONS / 3

Overview: Landau symbols

Notation	Definition	Analog to
$f(n) \in \mathcal{O}(g(n))$	see above	$\leq$
$f(n) \in o(g(n))$	see above	$<$
$f(n) \in \Omega(g(n))$	$g(n) \in \mathcal{O}(f(n))$	$\geq$
$f(n) \in \omega(g(n))$	$g(n) \in o(f(n))$	$>$
$f(n) \in \Theta(g(n))$	$f(n) \in \mathcal{O}(g(n))$ and $g(n) \in \mathcal{O}(f(n))$	$=$



Left panel $O(f(n))$, middle panel $\Omega(f(n))$ and right panel $\Theta(f(n))$

COMPLEXITY VS. EMPIRICAL RUNTIME

In this chapter we dealt with the **complexity of algorithms**:

- How does an algorithm **scale** with regards to the required resources?
- What happens when the problem gets bigger?
- What is the theoretical runtime complexity of an algorithm? (Knowledge / Estimation / Evidence)
  - Bubble sort has a worst-case runtime of $\mathcal{O}(n^2)$
  - Matrix multiplication of two regular $n \times n$ matrices has a runtime complexity of $\mathcal{O}(n^3)$
  - The Traveling Salesman Problem is NP-complete
  - ...
- It is often helpful to test the complexity of an algorithm empirically!

COMPLEXITY VS. EMPIRICAL RUNTIME / 2

But: How many resources does my algorithm **really** need?
→ **empirical runtime analysis**:

- Measurement of the runtime of an implementation on a given machine
- How much time (or memory etc.) is needed when the code is executed?
- → Depends on the machine, the compiler/interpreter, dependencies, and the code itself
- The empirical runtime can be measured for a fixed input quantity, but can also be systematically analyzed for different input quantities / problem instances
- When computing on a cluster, the cloud, or a machine on which several people are computing, the empirical run-time is usually influenced by the actions of other users