DS 320 Algorithms for Data Science **Lecture #1 Worksheet**
Fall 2024 Profs. Krzysztof Onak and Aldo Pacchiano

Important content included also in the introduction slides and syllabus!

Runtime Review

In runtime analysis we do an informal accounting. We count basic operations on values and variables of basic types (regular integers, booleans, single characters) as constant. Examples:

- comparing two integers
- reading value of a variable
- writing to a precomputed index in an array

Warning: In Python, default integers can be arbitrarily large, so operations on them have to depend on the size of their description—we will see examples of this later in the course. But as long as they are small, we can assume that they take constant time. Examples of "true" constant time integers for our purposes include `int32` in NumPy and `i32` in Rust.

Analyze the runtime of the following algorithm:

Algorithm 1 FindMinIndex($B[t+1, n]$).

Let MinIndex $= t + 1$.
for $i = t + 1$ to n **do**
  if $B[i] < B$[MinIndex] **then**
    MinIndex $= i$.
  end if
end for
return MinIndex.

Which lines of this pseudocode are constant-time?

Are there any loops? How many times do they run?

How do we combine these together to get the running time of the algorithm?

Which factors dominate asymptotically?

Asymptotic Notation

Definition 1 (Upper bound $O(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in O(g(n))$ if there exist ($\exists$) constants c_1, c_2 such that for all (s.t. $\forall$) $n \geq c_1$,

$$f(n) \leq c_2 g(n).$$

We'll often write $f(n) = O(g(n))$ because we are sloppy.

Translation: For large n (at least some c_1), the function $g(n)$ dominates $f(n)$ up to a constant factor.

Definition 2 (Lower bound $\Omega(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in \Omega(g(n))$ if there exist constants c_1, c_2 such that for all $n \geq c_1$,

$$f(n) \geq c_2 g(n).$$

Definition 3 (Tight bound $\Theta(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in \Theta(g(n))$ if $f \in O(g(n))$ and $f \in \Omega(g(n))$.

Exercise: True or False?

$f(n)$	$g(n)$	$O(g(n))$	$\Omega(g(n))$	$\Theta(g(n))$
$10^6 n^3 + 2n^2 - n + 10$	n^3			
$\sqrt{n} + \log n$	$\sqrt{n}$			
$n(\log n + \sqrt{n})$	$\sqrt{n}$			
n	n^2			

Definition 4 (Strict upper bound $o(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in o(g(n))$ if

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

or equivalently, for *any* constant $c_2 > 0$, there exists a constant c_1 such that for all $n \geq c_1$,

$$f(n) < c_2 g(n).$$

Definition 5 (Strict lower bound $\omega(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in \omega(g(n))$ if

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

or equivalently, for *any* constant $c_2 > 0$, there exists a constant c_1 such that for all $n \geq c_1$,

$$f(n) > c_2 g(n).$$

Asymptotic Properties

- Multiplication by a constant:
  If $f(n) = O(g(n))$ then for any $c > 0$, $c \cdot f(n) =$

- Transitivity:
  If $f(n) = O(h(n))$ and $h(n) = O(g(n))$ then $f(n) =$

- Symmetry:
  If $f(n) = O(g(n))$ then $g(n) =$
  If $f(n) = \Theta(g(n))$ then $g(n) =$

- Dominant Terms:
  If $f(n) = O(g(n))$ and $d(n) = O(e(n))$ then $f(n) + d(n) = O(\max\{g(n), e(n)\})$. It's fine to write this as $O(g(n) + e(n))$.

Common Functions

- Polynomials: $a_0 + a_1 n + \cdots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.
- Polynomial time: Running time is $O(n^d)$ for some constant d independent of the input size n.
- Logarithms: $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$. This means we can avoid specifying the base of the logarithm.
  For every $x > 0$, $\log n = o(n^x)$. Hence log grows slower than every polynomial.
- Exponentials: For all $r > 1$ and all $d > 0$, $n^d = o(r^n)$. Every polynomial grows slower than every exponential
- Factorial: By Sterling's formula, factorials grow faster than every exponential:

$$n! = (\sqrt{2\pi n}) \left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}.$$

Based on material developed by Prof. Kira Goldner