DS 320 Algorithms for Data Science **Lecture #1 Worksheet**
Spring 2025 Prof. Kira Goldner

Covered in introduction slides:

- Course policies (also in syllabus).
- Course learning objectives and what to expect in this class (also in FAQ).
- Sample of content we'll cover.

Announcement:

- Homework 0 on Gradescope due Thursday 11:59pm. Answer all the questions and get 100% toward participation.

Runtime Review

In runtime analysis we do an informal accounting. We count basic operations (algebra, array assignment, etc) as constant time.[1]

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?

[1] This isn't quite right—for example, multiplication of large numbers should scale with the bit complexity—but is a good approximation for us.

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			

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:

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