

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

When we analyze runtime, we'll do an informal accounting. We'll count basic operations (algebra, array assignment, etc) as constant time.<sup>1</sup>

We will 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  $B[i] < B[\text{MinIndex}]$  then  
        MinIndex =  $i$ .  
    end if  
end for  
return MinIndex.
```

---

Each of the following lines is a unit (constant-time) operation:

- **Let** MinIndex =  $t + 1$ .
- **if**  $B[i] < B[\text{MinIndex}]$  **then**
- MinIndex =  $i$ .

The for-loop runs  $n - t$  times (notice that both  $n$  and  $t$  are variables as they are in our input). Thus the runtime of this algorithm is  $O(n - t)$ .

---

<sup>1</sup>This isn't quite right—for example, multiplication of large numbers should scale with the bit complexity—but is a good approximation for us. We will analyze runtime by counting these operations.

# Asymptotic Notation

**Definition 1** (Upper bound  $O(\cdot)$ ). For a pair of functions  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ , 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.

Examples:

- $1 \in O(n)$ . This is because  $1 \leq 1 \cdot n$  (so  $c_2 = 1$ ) for all  $n \geq 1 = c_1$ .
- $n \in O(\frac{n}{2})$ . This is because  $n \leq 2 \cdot \frac{n}{2}$  (so  $c_2 = 2$ ) for all  $n \geq 1 = c_1$ .

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

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

Example:  $n \in \Omega(n + 7)$ . This is because  $n \geq \frac{1}{2} \cdot (n + 7)$  (so  $c_2 = \frac{1}{2}$ ) for all  $n \geq 7 = c_1$ .

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

**Exercise:** True or False?

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

Example solution: Let  $f(n) = 10^6 n^3 + 2n^2 - n + 10$ . For  $c_2 = (10^6 + 12)$ ,

$$10^6 n^3 + 2n^2 - n + 10 \leq c_2 n^3$$

for all  $n \geq 1$ , hence it is true that  $f(n) = O(n^3)$ .

For  $c_2 = 1$ ,  $10^6 n^3 + 2n^2 - n + 10 \leq c_2 n^3$ , hence it is true that it is  $f(n) = \Omega(n^3)$ .

Since  $f(n) = O(n^3)$  and  $f(n) = \Omega(n^3)$ , then  $f(n) = \Theta(n^3)$  as well.

## Asymptotic Properties

- Multiplication by a constant:

If  $f(n) = O(g(n))$  then for any  $c > 0$ ,  $c \cdot f(n) = O(g(n))$ .

- Transitivity:

If  $f(n) = O(h(n))$  and  $h(n) = O(g(n))$  then  $f(n) = O(g(n))$ .

- Symmetry:

If  $f(n) = O(g(n))$  then  $g(n) = \Omega(f(n))$ .

If  $f(n) = \Theta(g(n))$  then  $g(n) = \Theta(f(n))$ .

- 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:  $a_0 + a_1n + \dots + a_dn^d$  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:

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

## Strict Bounds

There are also strict bounds.

**Definition 4** (Strict upper bound  $o(\cdot)$ ). For a pair of functions  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ , we write  $f \in o(g(n))$  if 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  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ , we write  $f \in \omega(g(n))$  if for *any* constant  $c_2 > 0$ , there exists a constant  $c_1$  such that for all  $n \geq c_1$ ,

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