Intro: Using Big-O

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures and Algorithms
Algorithmic Toolbox

Learning Objectives

- Manipulate expressions involving Big-O and other asymptotic notation.
- Compute algorithm runtimes in terms of Big-O.

Big-O Notation

Definition

$f(n) = O(g(n))$ (f is Big-O of g) or $f \preceq g$ if there exist constants N and c so that for all $n \geq N$, $f(n) \leq c \cdot g(n)$.

Common Rules

Multiplicative constants can be omitted:

$$7n^3 = O(n^3),\ \frac{n^2}{3} = O(n^2)$$

Common Rules

Multiplicative constants can be omitted:

$$7n^3 = O(n^3),\ \frac{n^2}{3} = O(n^2)$$

$n^a \prec n^b$ for $0 < a < b$:

$$n = O(n^2),\ \sqrt{n} = O(n)$$

Common Rules

Multiplicative constants can be omitted:

$$7n^3 = O(n^3),\ \frac{n^2}{3} = O(n^2)$$

$n^a \prec n^b$ for $0 < a < b$:

$$n = O(n^2),\ \sqrt{n} = O(n)$$

$n^a \prec b^n$ $(a > 0, b > 1)$:

$$n^5 = O(\sqrt{2}^n),\ n^{100} = O(1.1^n)$$

Common Rules

Multiplicative constants can be omitted:

$$7n^3 = O(n^3),\ \frac{n^2}{3} = O(n^2)$$

$n^a \prec n^b$ for $0 < a < b$:

$$n = O(n^2),\ \sqrt{n} = O(n)$$

$n^a \prec b^n$ $(a > 0, b > 1)$:

$$n^5 = O(\sqrt{2}^n),\ n^{100} = O(1.1^n)$$

$(\log n)^a \prec n^b$ $(a, b > 0)$:

$$(\log n)^3 = O(\sqrt{n}),\ n \log n = O(n^2)$$

Common Rules

Multiplicative constants can be omitted:

$$7n^3 = O(n^3),\ \frac{n^2}{3} = O(n^2)$$

$n^a \prec n^b$ for $0 < a < b$:

$$n = O(n^2),\ \sqrt{n} = O(n)$$

$n^a \prec b^n$ $(a > 0, b > 1)$:

$$n^5 = O(\sqrt{2}^n),\ n^{100} = O(1.1^n)$$

$(\log n)^a \prec n^b$ $(a, b > 0)$:

$$(\log n)^3 = O(\sqrt{n}),\ n \log n = O(n^2)$$

Smaller terms can be omitted :

$$n^2 + n = O(n^2),\ 2^n + n^9 = O(2^n)$$

Recall Algorithm

Function FibList(n)

```
create an array F[0...n]
F[0] ← 0
F[1] ← 1
for i from 2 to n:
    F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Big-O in Practice

Operation	Runtime

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$
$F[1] \leftarrow 1$	$O(1)$

Big-O in Practice

Operation	Runtime
create an array $F[0 \dots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$
$F[1] \leftarrow 1$	$O(1)$
for i from 2 to n:	Loop $O(n)$ times

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$
$F[1] \leftarrow 1$	$O(1)$
for i from 2 to n:	Loop $O(n)$ times
$\quad F[i] \leftarrow F[i-1] + F[i-2]$	$O(n)$

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$
$F[1] \leftarrow 1$	$O(1)$
for i from 2 to n:	Loop $O(n)$ times
$\quad F[i] \leftarrow F[i-1] + F[i-2]$	$O(n)$
return $F[n]$	$O(1)$

Big-O in Practice

Operation	Runtime
create an array $F[0 \ldots n]$	$O(n)$
$F[0] \leftarrow 0$	$O(1)$
$F[1] \leftarrow 1$	$O(1)$
for i from 2 to n:	Loop $O(n)$ times
$\quad F[i] \leftarrow F[i-1] + F[i-2]$	$O(n)$
return $F[n]$	$O(1)$

Total:

$$O(n)+O(1)+O(1)+O(n)\cdot O(n)+O(1) = O(n^2).$$

Other Notation

Definition

For functions $f, g : \mathbb{N} \to \mathbb{R}^+$ we say that:

- $f(n) = \Omega(g(n))$ or $f \succeq g$ if for some c, $f(n) \geq c \cdot g(n)$ (f grows no slower than g).
- $f(n) = \Theta(g(n))$ or $f \asymp g$ if $f = O(g)$ and $f = \Omega(g)$ (f grows at the same rate as g).

Other Notation

Definition

For functions $f, g : \mathbb{N} \to \mathbb{R}^+$ we say that:

- $f(n) = o(g(n))$ or $f \prec g$ if $f(n)/g(n) \to 0$ as $n \to \infty$ (f grows slower than g).

Asymptotic Notation

- Lets us ignore messy details in analysis.
- Produces clean answers.
- Throws away a lot of practically useful information.