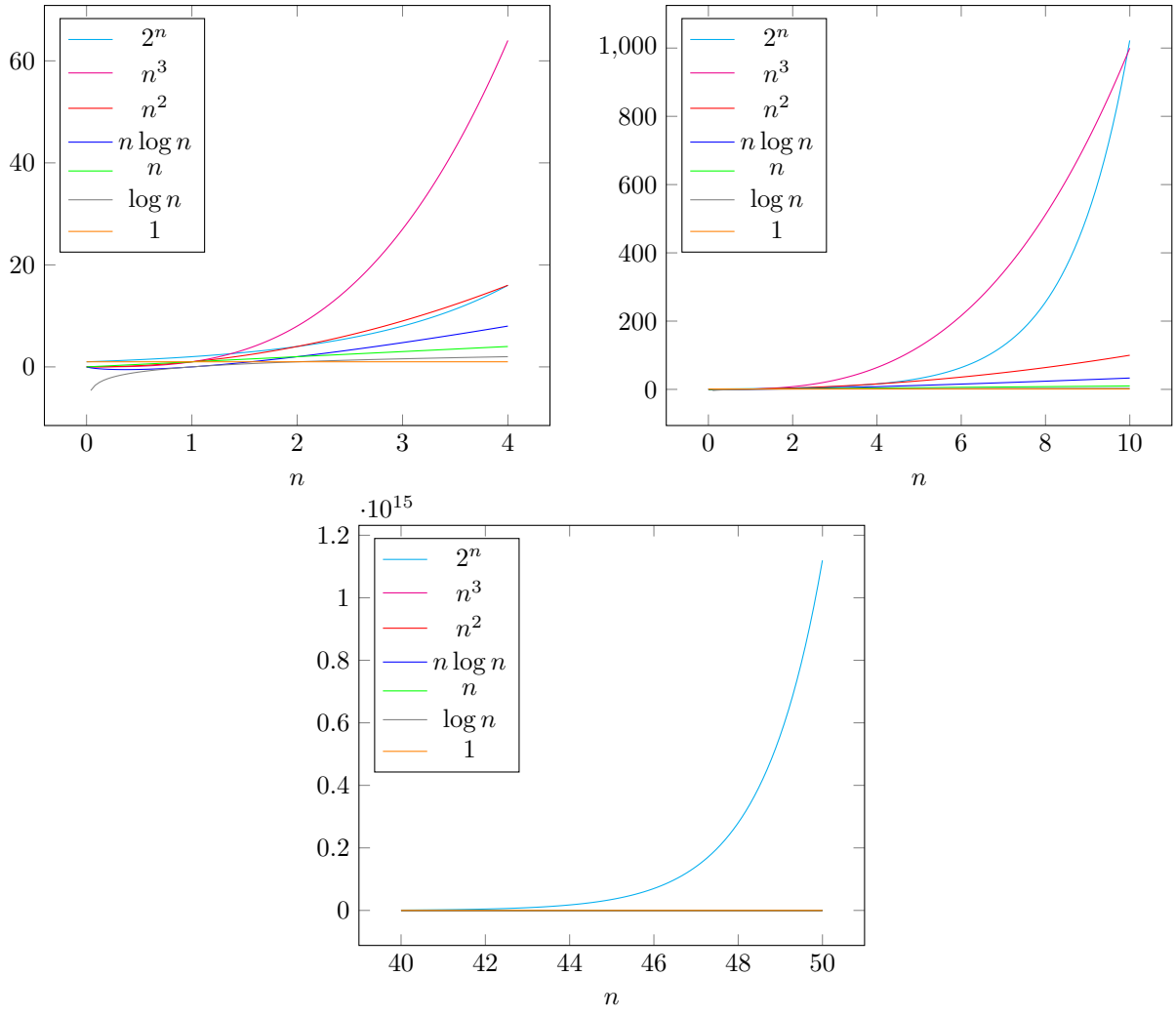
Quick Guide to Big-$\mathcal{O}$ Values

Emilie Menard Barnard
Foothill College

Use this pdf as a quick-reference guide for everything Big-$\mathcal{O}$.

Charts of Common Values



Big-$\mathcal{O}$ focuses on what happens to the algorithm in the worst-case when n grows really large. Take note of the domain of the n values on the x-axes. As $n \to \infty$, we have:

$$\boxed{2^n > n^3 > n^2 > n \log n > n > \log n > 1}$$

Table of Common Algorithms & Code Features and Their Big-$\mathcal{O}$ Runtimes

Algorithms & Code Features	Big-$\mathcal{O}$ Runtime
single-line math operations, determining if a binary integer is even or odd	$\mathcal{O}(1)$ "constant runtime"
binary search	$\mathcal{O}(\log n)$ "logarithmic runtime"
one loop, searching an unsorted array	$\mathcal{O}(n)$ "linear runtime"
some sorting algorithms*	$\mathcal{O}(n \log n)$
two nested loops, addition of two $n \times n$ matrices	$\mathcal{O}(n^2)$ "quadratic runtime"
three nested loops, multiplication of two $n \times n$ matrices	$\mathcal{O}(n^3)$ "cubic runtime"
recursive Fibonacci algorithm, some growth rate algorithms	$\mathcal{O}(2^n)$ "exponential runtime"

*Next week, we'll talk more about sorting. Some sorting algorithms are not $\mathcal{O}(n \log n)$, but for now you can assume most are. I will provide an updated table once we cover sorting more in-depth.

Generalized Big-$\mathcal{O}$ Calculation Rules

- Runtimes of pieces of code that come one after another (i.e. the first piece of code finishes before the second begins) are *added*
  Example:

```
for (int i = 0; i < n; i++ ) {
     cout << i << endl;
}
for (int j = 0; j < n; j++ ) {
     cout << j << endl;
}
```

  Runtime is $\boldsymbol{\mathcal{O}(n+n)} = \mathcal{O}(2n) = \mathcal{O}(n)$
  (follow rules below to simplify from the bold runtime above)

- Runtimes of pieces of code that are nested (i.e. the first piece of code is still going before the second begins) are *multiplied*
  Example:

```
for (int i = 0; i < n; i++ ) {
     for (int j = 0; j < n; j++ ) {
          cout << i + j << endl;
     }
}
```

  Runtime is $\boldsymbol{\mathcal{O}(n \cdot n)} = \mathcal{O}(n^2)$

- Coefficients should be dropped
  Examples:
  - $\mathcal{O}(2n) \rightarrow \mathcal{O}(n)$

  - $\mathcal{O}(4n \log n) \to \mathcal{O}(n \log n)$
  - But $\mathcal{O}(2^n)$ stays as $\mathcal{O}(2^n)$

- Keep only the highest-order or "largest" term
  Examples:
  - $\mathcal{O}(n^3 + n^2 + 1) \to \mathcal{O}(n^3)$
  - $\mathcal{O}(n \log n + n) \to \mathcal{O}(n \log n)$