CIS 121—Data Structures and Algorithms with Java - Fall 2015

Big-Oh Lab — TBD

Learning Goals

During this lab, you will:

- review Bachmann-Landau notation
- examine certain functions and their relative asymptotic growth rates
- examine the runtime complexity of code
- prove Bachmann-Landau relations

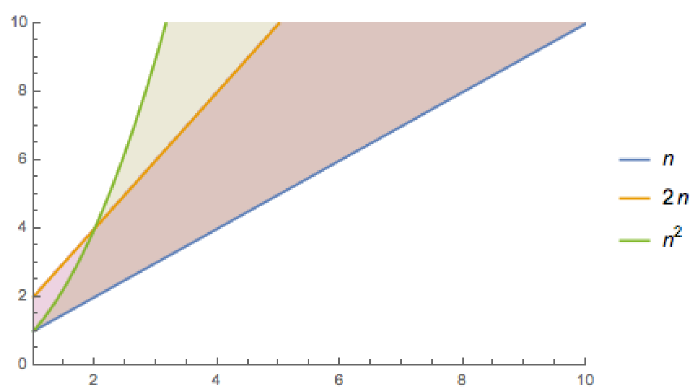
Big-Oh and Bachmann-Landau Notation

In class, you have started to discuss Big Oh and other ways of classifying functions and algorithms. These notations belong to what is commonly referred to as the *Bachmann-Landau* family of notations.

Big-Oh Notation

Definition. $f(n) = O(g(n))$ if there exist constants n_0 and $c > 0$ s.t. $f(i) \leq cg(i)$ for all $i \geq n_0$.

Simplified: If $f(n)$ is $O(g(n))$, $g(n)$ is an asymptotic upper bound for $f(n)$.

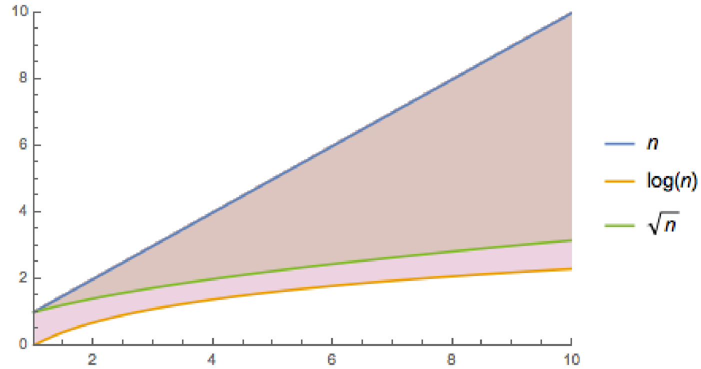


$n = O(n)$, $n = O(n^2)$

Big-Omega Notation

Definition. $f(n) = \Omega(g(n))$ if there exist constants n_0 and $c > 0$ s.t. $f(i) \geq cg(i)$ for all $i \geq n_0$.

Simplified: If $f(n)$ is $\Omega(g(n))$, $g(n)$ is an asymptotic lower bound for $f(n)$.

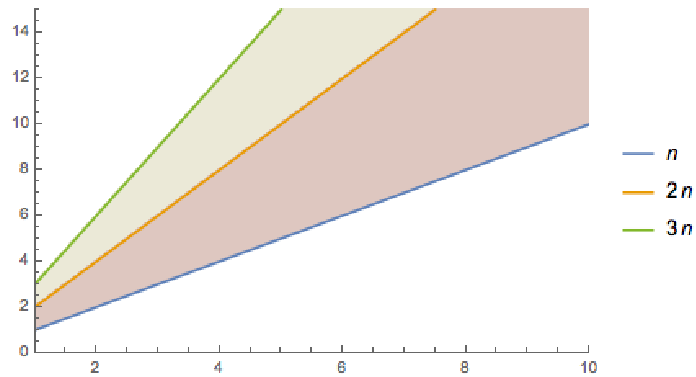


$n = \Omega(\lg n)$, $n = \Omega(\sqrt{n})$

Big-Theta Notation

Definition. $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Simplified: If $f(n)$ is $\Theta(g(n))$, $g(n)$ is an asymptotic *tight* bound for $f(n)$.



$n = \Theta(n)$. Every function is Big-Θ of itself.

As a protip, it is also good to note that the Bachmann-Landau notations refer to *classes of functions*. When you read $f(n) = O(g(n))$, this is equivalent to the statement:

$$f(n) \in O(g(n))$$

. Specifically, $f(n)$ is in the class of functions which are asymptotically bounded above by $g(n)$. Likewise, Big-Ω and Big-Θ both reflect classes of functions.

Stirling's Approximation

$$n! \sim (\frac{n}{e})^n \sqrt{2\pi n}$$

Lab Problems

Problem 1

Order the following functions such that if f precedes g, then $f(n)$ is $O(g(n))$.

$$\sqrt{n},\ n,\ n^{1.5},\ n^2,\ n \lg n,\ n \lg\lg n,\ n \lg n^2,\ 2^{n/2},\ 2^n,\ \lg(n!),\ n^2 \lg n,\ n^3,\ 2^{2^n}$$

Problem 2

Provide a runtime analysis of the following loop:

```
for(int i = 0; i < n; i++)
    for (int j = i; j <= n; j++)
        for (int k = i; k <= j; k++)
            sum++;
```

Problem 3

In this problem, you are **not** allowed to use the theorems about Big-Oh stated in the lecture notes. Your proof should follow exclusively from the definition of Big-Oh.

Prove or disprove the following statement:

$$f(n) + g(n) \text{ is } \Theta(\max\left\{f(n), g(n)\right\}), \text{ where } f, g : R \to R^+.$$

Problem 4

Prove or disprove the following statement:

$$2^n \text{ is } O(n!).$$

Problem 5

Provide a runtime analysis of the following loop:

```
for (int i = 2; i < n; i = i*i)
    for (int j = 1; j < Math.sqrt(i); j = j+j)
        System.out.println("*");
```

Problem 6

Prove or disprove the following statement:

$$\lg(n!) \text{ is } \Theta(n \lg n).$$

Logarithms Cheat Sheet

Exponential terms appear very frequently in the study of algorithms and their runtimes. Therefore, logarithms are very useful when manipulating exponential terms in Big-Oh proofs! It is therefore advised that you become very familiar with logs.

Here's a little cheat-sheet for you to refresh your memory!

Properties of Logarithms

$$\log(c \cdot f(n)) = \log c + \log f(n)$$

$$\log x^y = y \log x$$

$$\log a + \log b = \log(ab)$$

$$\log a - \log b = \log(a/b)$$

$$\log_b x = \frac{\log_c x}{\log_c b}$$

$$\log 2^n = n$$

$$\log n! = \log[n \cdot (n-1) ... 2 \cdot 1] = \log n + \log(n-1) + ... + \log 1 = \sum_{i=1}^{n} \log i$$