CS240

Nathan Sprague

January 24, 2020

Alternate Definition of Big-O

Big O

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

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c < \infty$$

where c is some constant (possibly 0)

Alternate Definitions of Big-O

Big O

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$$

where c is some constant (possibly 0)

- $n^3 + 2n \in n^3$

Alternate Definitions of Big-O

Big O

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

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c < \infty$$

where c is some constant (possibly 0)

- $n^3 + 2n \in n^3$
- $$\lim_{n \to \infty} \frac{n^3 + 2n}{n^3} = \lim_{n \to \infty} 1 + \frac{2}{n^2} = 1$$

Alternate Definitions of O, Ω, Θ

Big Ω

$f(n) \in \Omega(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0$$

where c is some constant (possibly ∞)

Alternate Definitions of O, Ω, Θ

Big Θ

$f(n) \in \Theta(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c, 0 < c < \infty$$

where c is some constant.

A Complication

- Let's analyze this algorithm:

```
public static boolean contains(int target,
                               int[] numbers) {
  for (int number : numbers) {
    if (number == target) {
      return true;
    }
  }
  return false;
}
```

Best, Worst, Average Case

```java
public static boolean contains(int target,
                               int[] numbers) {
  for (int number : numbers) {
    if (number == target) {
      return true;
    }
  }
  return false;
}
```

- Best Case: 1 comparison, $O(1)$
- Worst Case: n comparisons, $O(n)$
- Average Case: $\frac{n+1}{2}$ comparisons, $O(n)$

Refined Algorithm Analysis Algorithm

- STEP 1: Decide on best, worst, or average case analysis
- STEP 2: Select a measure of input size and a basic operation
- STEP 3: Find a function $T(n)$ that describes the number of times the basic operation occurs
- STEP 4: Describe $T(n)$ using order notation:
  - Big-O for an upper bound
    "The algorithm is at least this fast!"
  - Big-Ω for a lower bound
    "The algorithm is at least this slow!"
  - Big-Θ for both upper and lower bound

L'Hôpital's Rule

L'Hôpital's Rule

If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ and $f'(n)$ and $g'(n)$ exist, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

L'Hôpital Example

- $n \log_2 n \overset{?}{\in} O(n^2)$

L'Hôpital Example

- $n \log_2 n \stackrel{?}{\in} O(n^2)$
- $\lim_{n \to \infty} \frac{n \log_2 n}{n^2} = \lim_{n \to \infty} \frac{\log_2 n}{n}$

L'Hôpital Example

- $n \log_2 n \stackrel{?}{\in} O(n^2)$
- $\lim\limits_{n \to \infty} \dfrac{n \log_2 n}{n^2} = \lim\limits_{n \to \infty} \dfrac{\log_2 n}{n}$
- $= \lim\limits_{n \to \infty} \dfrac{\ln n}{n \ln 2}$ (Recall that $\log_b(n) = \dfrac{\log_k n}{\log_k b}$)

L'Hôpital Example

- $n \log_2 n \stackrel{?}{\in} O(n^2)$
- $\lim_{n\to\infty} \frac{n \log_2 n}{n^2} = \lim_{n\to\infty} \frac{\log_2 n}{n}$
- $= \lim_{n\to\infty} \frac{\ln n}{n \ln 2}$ (Recall that $\log_b(n) = \frac{\log_k n}{\log_k b}$)
- Apply L'Hôpital's rule:
- $= \lim_{n\to\infty} \frac{\frac{1}{n}}{\ln 2}$ (Recall that $\frac{d}{dx} \ln x = 1/x$)

L'Hôpital Example

- $n \log_2 n \stackrel{?}{\in} O(n^2)$
- $\lim_{n \to \infty} \frac{n \log_2 n}{n^2} = \lim_{n \to \infty} \frac{\log_2 n}{n}$
- $= \lim_{n \to \infty} \frac{\ln n}{n \ln 2}$ (Recall that $\log_b(n) = \frac{\log_k n}{\log_k b}$)
- Apply L'Hôpital's rule:
- $= \lim_{n \to \infty} \frac{\frac{1}{n}}{\ln 2}$ (Recall that $\frac{d}{dx} \ln x = 1/x$)
- $= \lim_{n \to \infty} \frac{1}{n \ln 2} = 0$

What If We Want to Show That f(n) is NOT O(g(n))

- Easiest approach is usually to show:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

OpenDSA Question

Suppose that a particular algorithm has time complexity $T(n) = 3 \times 2^n$ and that executing an implementation of it on a particular machine takes t seconds for n inputs. Now suppose that we are presented with a machine that is 64 times as fast. How many inputs could we process on the new machine in t seconds?

OpenDSA Question

- Let's call the input size we could handle before n_{old}. The number of steps we completed in t seconds was: $3 \times 2^{n_{old}}$.
- Since our new computer is 64 times faster, the number of steps we can perform in t seconds is now $64 \times 3 \times 2^{n_{old}}$
- Our complexity function tells us that $steps = 3 \times 2^n$, we can solve for size (n):
- $s = 3 \times 2^n$
- $s/3 = 2^n$
- $\log_2(s/3) = n$
- $n = \log_2(s/3)$

OpenDSA Question

Now we plug in our step budget for s:

- $n = \log_2(\frac{64 \times 3 \times 2^{n_{old}}}{3})$
- $n = \log_2(64 \times 2^{n_{old}})$
- $n = \log_2(2^6 \times 2^{n_{old}})$
- $n = \log_2(2^{n_{old}+6})$
- $n = n_{old} + 6$