CS240

Nathan Sprague

September 12, 2014

Algorithm Analysis

- What should we measure?
  - clarity/simplicity?
  - space efficiency?
  - time efficiency?
- A story...

Take-Home Messages (1/2)

- Analysis must account for input size
  - How does the running time change as the input size increases?

Take-Home Messages (2/2)

- Goal is to analyze algorithms, not programs.
- Running time of programs is subject to:
    - Programming language
    - Speed of the computer
    - Computer load
    - Compiler version
    - ...

If Not Time, Then What?

If Not Time, Then What?

- Number of steps that the algorithm takes to complete
- Goal: develop a function that maps from input size to the number of steps
- Examples...

```
sum = 0
for num in numbers:
    sum += num
```

```
sum = 0
for num1 in numbers:
    for num2 in numbers:
        sum += 1
```

Basic Operations

- Goal restated: develop a function that maps from input size to the number of times the "basic operation" is performed
- No single correct choice for basic operation
- Guideline:
  - Should happen in inner-most loop
- If chosen well, count will be proportional to execution time

Growth/Complexity Functions

- Let's look at them...
- Goal restated: Map our algorithm to a complexity function

Big-O

- Informal description: Growth functions are categorized according to their dominant (fastest growing) term
- Constants and lower-order terms are discarded
- Examples:
  - $10n \in O(n)$
  - $5n^2 + 2n + 3 \in O(n^2)$
  - $n \log n + n \in O(n \log n)$

Why Drop the Constants?

- Example...

Why Drop the Constants?

- Despite constants, functions from slower growing classes will always be faster eventually
- Real goal is to understand the relative impact of increasing input size
- Side benefit: justifies flexibility in choosing basic operation

Why Drop Lower Order Terms

- Contribution of lower-order terms becomes insignificant as input size increases
- Example...

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

- Informal rule of "dropping constants" follows immediately:
  - $50n \stackrel{?}{\in} O(n)$

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

- Informal rule of "dropping constants" follows immediately:
  - $50n \overset{?}{\in} O(n)$
  - Yes! choose $c = 50, n_0 = 1$, clearly
  - $50n \leq 50n$ for all $n \geq 1$

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

- Informal rule of "dropping lower-order terms" also follows:
  - $n^2 + 40n \overset{?}{\in} O(n^2)$

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

- Informal rule of "dropping lower-order terms" also follows:
  - $n^2 + 40n \stackrel{?}{\in} O(n^2)$
  - Notice that:
    $n^2 + 40n \leq n^2 + 40n^2 = 41n^2$

Formal Definition of Big-O

Big O

$f(n) \in O(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

- Informal rule of "dropping lower-order terms" also follows:
  - $n^2 + 40n \overset{?}{\in} O(n^2)$
  - Notice that: $n^2 + 40n \leq n^2 + 40n^2 = 41n^2$
  - Choose $c = 41$, $n_0 = 1$, clearly $n^2 + 40n \leq 41n^2$ for all $n \geq 1$

Algorithm Analysis Algorithm

- STEP 1: Select a measure of input size and a basic operation
- STEP 2: Develop a function $T(n)$ that describes the number of times the basic operation occurs as a function of input size
- STEP 3: Describe $T(n)$ using order notation (Big-O)

Big O Describes an Upper Bound

- Big O is loosely analogous to $\leq$
- All of these statements are true:
  $n^2 \in O(n^2)$
  $n^2 \in O(n^4)$
  $n^2 \in O(n!)$
  ...
  $2n^2 \in O(n^2)$

Big Omega

Big Ω

$f(n) \in \Omega(g(n))$ iff there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for all $n \geq n_0$,

$$f(n) \geq cg(n)$$

- Big Ω is loosely analogous to $\geq$
- All of these statements are true:
  $n^2 \in \Omega(n^2)$
  $n^4 \in \Omega(n^2)$
  $n! \in \Omega(n^2)$
  ...
  $n^2 \in \Omega(2n^2)$

Big Theta

Big Θ

$f(n) \in \theta(g(n))$ iff,

$$f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n))$$

- Big Θ is loosely analogous to $=$
- Which of these statements are true?

  $n^2 \stackrel{?}{\in} \Theta(n^2)$

  $2n^2 \stackrel{?}{\in} \Theta(n^2)$

  $n^2 \stackrel{?}{\in} \Theta(n^4)$

  $5n^2 + 2n \stackrel{?}{\in} \Theta(4n^3)$

Big Theta

Big Θ

$f(n) \in \theta(g(n))$ iff,

$$f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n))$$

- Big Θ is loosely analogous to $=$
- Which of these statements are true?
  $n^2 \in \Theta(n^2)$
  $2n^2 \in \Theta(n^2)$
  $n^2 \notin \Theta(n^4)$
  $5n^2 + 2n \notin \Theta(4n^3)$

Alternate Definitions of O, Ω, Θ

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 O, Ω, Θ

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 O, Ω, Θ

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)

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

$$\lim_{n\to\infty} \frac{n^3+2n}{n^3} = \lim_{n\to\infty} \left(1 + \frac{2}{n^2}\right) = \lim_{n\to\infty} 1 + \lim_{n\to\infty} \left(\frac{2}{n^2}\right) = 1$$

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:

```python
def contains(key, numbers):
    for num in numbers:
        if key == num:
            return True
    return False
```

Best, Worst, Average Case

```python
def contains(key, numbers):
    for num in numbers:
        if key == num:
            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

Quiz (1)

- Input size?
- Operation to count??
- Growth function?
- Big - Θ?

```python
def some_func(values):
    sum = 0
    for i in values:
        sum += i
    for i in range(20):
        sum += i
    return sum
```

Quiz (2)

- Input size?
- Operation to count??
- Growth function?
- Big - Θ?

```
def some_func(values):
    sum = 0
    for i in values:
        sum += i
        for j in range(20):
            sum += j
    return sum
```

Quiz (3)

- Input size?
- Operation to count??
- Growth function?
- Big - Θ?

```
def some_func(values):
    sum = 0
    indx = 1
    while indx <= len(values):
        sum += values[indx - 1]
        indx *= 2
    return sum
```

Quiz (4)

```
def some_func(values):
    sum = 0

    for i in range(1000):
        sum = sum + i

    for num in values:
        indx = 1
        while indx <= len(values):
            sum += values[indx - 1]
            indx *= 2

    return sum
```

Quiz (1)

- Input size? len(values)
- Operation to count?? +
- Growth function? $f(n) = n+20$
- Big - Θ? $n+20 \in \Theta(n)$

```
def some_func(values):
    sum = 0
    for i in values:
        sum += i
    for i in range(20):
        sum += i
    return sum
```

n 1 20 1 $n+20$

Quiz (2)

- Input size? len(values)
- Operation to count?? +
- Growth function? $f(n) = 21n$
- Big - Θ? $21n \in \Theta(n)$

```
def some_func(values):
    sum = 0
    for i in values:
        sum += i
        for j in range(20):
            sum += j
    return sum
```

n, 1, 20, 1

$n + 20n = 21n$

Quiz (3)

- Input size? len(values)
- Operation to count??+
- Growth function? $f(n) = \log_2 n + 1$
- Big - Θ? $\Theta(\log n)$

$f(1) = 1$
$f(2) = 2$
$f(4) = 3$
$f(8) = 4$
$\vdots$
$f(n) = \log_2 n + 1$

```
def some_func(values):
    sum = 0
    indx = 1
    while indx <= len(values):
      1 sum += values[indx - 1]
        indx *= 2
    return sum
```

Quiz (4)

```
def some_func(values):
    sum = 0

    for i in range(1000):
        sum = sum + i

    for num in values:
        indx = 1
        while indx <= len(values):
            sum += values[indx - 1]
            indx *= 2

    return sum
```

1000

1

n

1

$\log_2 n + 1$

$1{,}000 + n(\log_2 n + 1)$

$= n\log_2 n + n + 1000$

$\in \Theta(n \log n)$

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

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

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

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

L'Hôpital Example

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

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

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