CSE 373: Data Structures and Algorithms

Lecture 4: Math Review/Asymptotic Analysis II

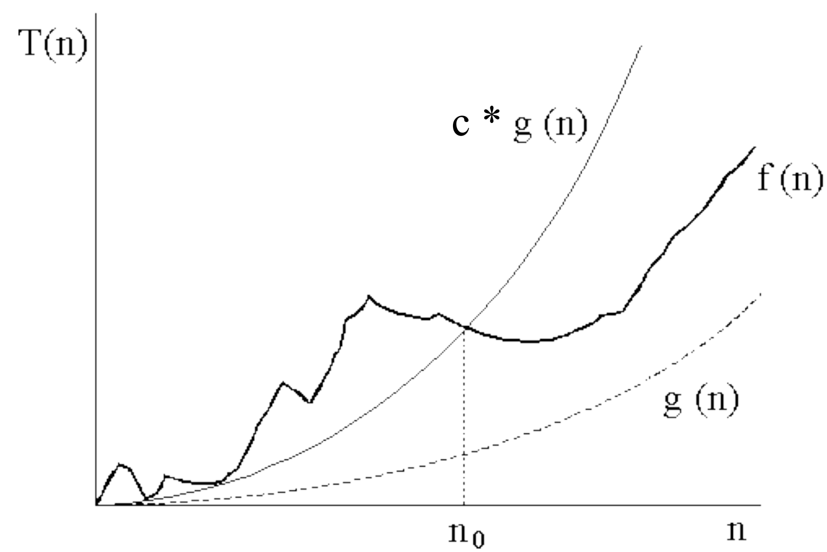
Big-Oh notation

- Asymptotic upper bound

- Defn: $f(n) = O(g(n))$, if there exists positive constants c , n_0 such that: $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

- Idea: We are concerned with how the function grows when N is large. We are not concerned with constant factors: coarse distinctions among functions

- Lingo: "$f(n)$ grows no faster than $g(n)$."



Functions in Algorithm Analysis

- $f(n) : \{0, 1, \ldots\} \rightarrow \Re^+$
  - domain of *f* is the nonnegative integers
  - range of *f* is the nonnegative reals

- Unless otherwise indicated, the symbols *f*, *g*, *h*, and *T* refer to functions with this domain and range.

- We use many functions with other domains and ranges.
  - Example: $f(n) = 5\, n \log_2 (n/3)$
    - Although the domain of *f* is nonnegative integers, the domain of $\log_2$ is all positive reals.

Big-Oh example problems

- $n = O(2n)$?
- $2n = O(n)$?
- $n = O(n^2)$?
- $n^2 = O(n)$?
- $n = O(1)$?
- $100 = O(n)$?
- $214n + 34 = O(2n^2 + 8n)$?

Preferred big-Oh usage

- pick tightest bound. If $f(n) = 5n$, then:

  $f(n) = O(n^5)$
  $f(n) = O(n^3)$
  $f(n) = O(n \log n)$
  $f(n) = O(n)$ ← preferred

- ignore constant factors and low order terms

  $f(n) = O(n)$, *not* $f(n) = O(5n)$
  $f(n) = O(n^3)$, *not* $f(n) = O(n^3 + n^2 + n \log n)$

  – Wrong: $f(n) \leq O(g(n))$
  – Wrong: $f(n) \geq O(g(n))$

Show f(n) = O(n)

Claim: $2n + 6 = O(n)$

Proof: Must find c, n_0 such that for all $n > n_0$,

$2n + 6 <= n$

Big omega, theta

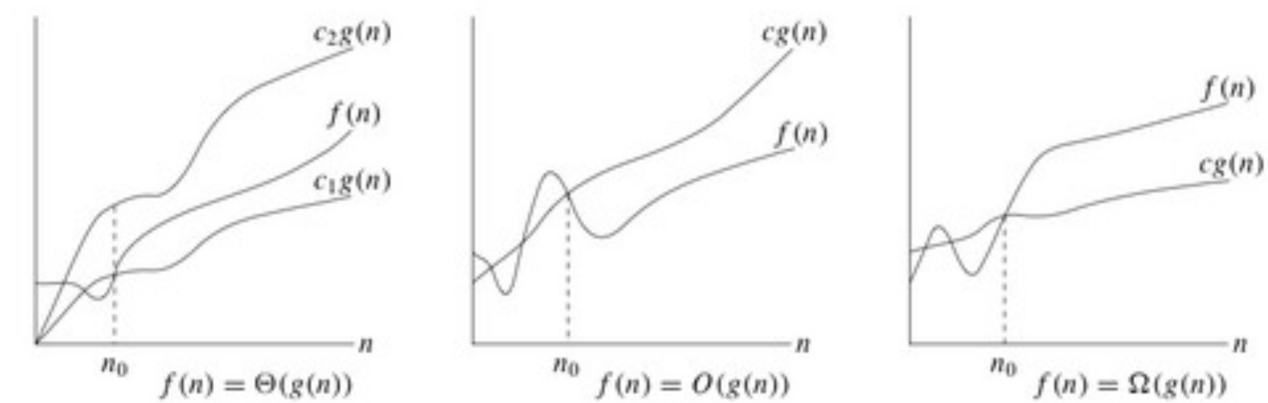
- **big-Oh Defn**: $f(n) = O(g(n))$ if there exist positive constants c , n_0 such that:
  $f(n) \le c \cdot g(n)$ for all $n \ge n_0$

- **big-Omega Defn**: $f(n) = \Omega(g(n))$ if there are positive constants c and n_0 such that $f(n) \ge c\, g(n)$ for all $n \ge n_0$
  - Lingo: "$f(n)$ grows no slower than $g(n)$."

- **big-Theta Defn**: $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
  - Big-Oh, Omega, and Theta establish a *relative ordering* among all functions of n

Intuition about the notations

notation	intuition
O (Big-Oh)	$f(n) \leq g(n)$
Ω (Big-Omega)	$f(n) \geq g(n)$
Θ (Theta)	$f(n) = g(n)$



Efficiency examples 3

```
sum = 0;
for (int i = 1; i <= N * N; i++) {
    for (int j = 1; j <= N * N * N; j++) {
        sum++;
    }
}
```

? ? ?

Efficiency examples 3

```
sum = 0;
for (int i = 1; i <= N * N; i++) {
    for (int j = 1; j <= N * N * N; j++) {
        sum++;
    }
}
```

N^3 N^2 $N^5 + 1$

- So what is the Big-Oh?

Math background: Exponents

- Exponents
  - X^Y, or "X to the Y^{th} power";
    X multiplied by itself Y times

- Some useful identities
  - $X^A X^B = X^{A+B}$
  - $X^A / X^B = X^{A-B}$
  - $(X^A)^B = X^{AB}$
  - $X^N + X^N = 2X^N$
  - $2^N + 2^N = 2^{N+1}$

Efficiency examples 4

```
sum = 0;
for (int i = 1; i <= N; i += c) {
    sum++;
}
```

?

?

Efficiency examples 4

```
sum = 0;
for (int i = 1; i <= N; i += c) {
   sum++;
}
```

N/c

N/c + 1

- What is the Big-Oh?
  - Intuition: Adding to the loop counter means that the loop runtime grows linearly when compared to its maximum value *n*.

Efficiency examples 5

```
sum = 0;
for (int i = 1; i <= N; i *= c) {
   sum++;
}
```

? ?

- Intuition: Multiplying the loop counter means that the maximum value *n* must grow exponentially to linearly increase the loop runtime

Efficiency examples 5

```
sum = 0;
for (int i = 1; i <= N; i *= c) {
   sum++;
}
```

$\log_c N$

$\log_c N + 1$

- What is the Big-Oh?

Math background: Logarithms

- Logarithms
  - *definition*: $X^A = B$ if and only if $\log_X B = A$
  - *intuition*: $\log_X B$ means:
    "the power X must be raised to, to get B"
  - In this course, a logarithm with no base implies base 2.
    $\log B$ means $\log_2 B$
- Examples
  - $\log_2 16 = 4$ (because $2^4 = 16$)
  - $\log_{10} 1000 = 3$ (because $10^3 = 1000$)

Logarithm identities

Identities for logs with addition, multiplication, powers:

- log (AB) = log A + log B
- log (A/B) = log A – log B
- log (A^B) = B log A

Identity for converting bases of a logarithm:

- $$\log_A B = \frac{\log_C B}{\log_C A} \quad A, B, C > 0, A \neq 1$$
  - example:
    $\log_4 32 = (\log_2 32) / (\log_2 4)$
    $= 5 / 2$

Techniques: Logarithm problem solving

- When presented with an expression of the form:
  - $\log_a X = Y$

  and trying to solve for X, raise both sides to the a power.
  - $X = a^Y$

- When presented with an expression of the form:
  - $\log_a X = \log_b Y$

  and trying to solve for X, find a common base between the logarithms using the identity on the last slide.
  - $\log_a X = \log_a Y / \log_a b$

Logarithm practice problems

- Determine the value of *x* in the following equation.
  - $\log_7 x + \log_7 13 = 3$

- Determine the value of *x* in the following equation.
  - $\log_8 4 - \log_8 x = \log_8 5 + \log_{16} 6$

Prove identity for converting bases

Prove $\log_a b = \log_c b / \log_c a$.

A log is a log…

- We will assume all logs are to base 2

- Fine for Big Oh analysis because the log to one base is equivalent to the log of another base within a constant factor
  - E.g., $\log_{10}x$ is equivalent to $\log_{2}x$ within what constant factor?

Efficiency examples 6

```
int sum = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i / 2; j += 2) {
        sum++;
    }
}
```

Math background: Arithmetic series

- Series

$$\sum_{i=j}^{k} Expr$$

  – for some expression *Expr* (possibly containing *i*), means the sum of all values of *Expr* with each value of *i* between *j* and *k* inclusive

Example:

$$\sum_{i=0}^{4} 2i + 1$$

= (2(0) + 1) + (2(1) + 1) + (2(2) + 1)
  + (2(3) + 1) + (2(4) + 1)
= 1 + 3 + 5 + 7 + 9
= 25

Series identities

- sum from 1 through N inclusive

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

- is there an intuition for this identity?
  - sum of all numbers from 1 to N

    1 + 2 + 3 + ... + (N-2) + (N-1) + N

  - how many terms are in this sum? Can we rearrange them?