Announcements

HW2 has been posted in the Blackboard and class website

Due on Thursday, Feb 3 before class starts.

Asymptotic Growth Rate

A way of comparing functions that ignores constant factors and small input sizes

$O(g(n))$: **class of functions $f(n)$ that grow *__no faster__* than $g(n)$ → order or growth of $g(n)$ $\geq$ order or growth of $f(n)$**

$\Theta\ (g(n))$: **class of functions $f(n)$ that grow *__at same rate__* as $g(n)$ → order or growth of $g(n) =$ order or growth of $f(n)$**

→ $f(n)$ has an efficiency class of $g(n)$

$\Omega(g(n))$: **class of functions $f(n)$ that grow *__at least as fast__* as $g(n)$**

Establishing rate of growth

- By definition: There exist positive constant **c** and non-negative integer **n_0** such that

$$f(n) \in O(g(n)) \quad \text{if} \quad f(n) \le cg(n) \quad n \ge n_0$$

Establishing rate of growth

- By limits:

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

- 0: order of growth of $f(n)$ < order of growth of $g(n)$ — $f(n) \in O(g(n))$
- $c \neq 0$: order of growth of $f(n)$ = order of growth of $g(n)$ — $f(n) \in O(g(n)), \Theta(g(n))$, and $\Omega(g(n))$
- ∞: order of growth of $f(n)$ > order of growth of $g(n)$ — $f(n) \in \Omega(g(n))$

L'Hôpital's Rule:

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

The derivatives $f'(n)$ **and** $g'(n)$ **exist,**

Then

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

Examples

Compare the functions

Steps:

1. Establish $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

2. Simplify the ratio $\frac{f(n)}{g(n)}$

3. Apply L'Hôpital's Rule

n vs. $\log^2 n$

$n \log n$ vs. $\sqrt{n^3}$

$n!$ vs. n^n

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \quad \text{(Stirling's formula)}$$

Some Important Properties of Order of Growth

- **All logarithmic functions $log_a n$ belong to the same class $\Theta(\log n)$ no matter what the logarithm's base $a > 1$ is**

- **All polynomials of the same degree k belong to the same class:**

$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 \in \Theta(n^k)$$

- **Exponential functions a^n have different orders of growth for different a's**

- **order $\log n$ < order n^α $(\alpha > 0)$ < order a^n < order $n!$ < order n^n**

Reading Assignments

Review how to calculate the derivative for simple functions from your Calculus classes

Chapter 2.2-2.3 and Appendix A

Analyze the Time Efficiency of An Algorithm

Nonrecursive Algorithm

- Matrix multiplication
- Selection sort
- etc

ALGORITHM $Factorial(n)$

$f \leftarrow 1$

for $i \leftarrow 1$ **to** n **do**

  $f \leftarrow f * i$

return f

Recursive Algorithm

- Fibonacci number
- Merge sort
- etc

ALGORITHM $Factorial(n)$

if $n = 0$

  return 1

else

  return $Factorial(n-1) * n$

Analyze the Time Efficiency of A Nonrecursive Algorithm

ALGORITHM $Factorial(n)$

$f \leftarrow 1$

for $i \leftarrow 1$ **to** n **do**

    $f \leftarrow f * i$

return f

Input size?

Basic operations?

Worst case, best case, and average case

Time efficiency of Nonrecursive Algorithms

Steps in mathematical analysis of nonrecursive algorithms:

- Decide on parameter *n* indicating *input size*
- Identify algorithm's *basic operation*
- Determine *worst*, *average*, and *best* case for input of size *n*
- Set up summation for *t(n)* reflecting algorithm's loop structure
- Simplify summation using standard formulas (see Appendix A)

Useful Formulas in Appendix A

$$\sum_{i=startInd}^{endInd} 1 = \underbrace{1 + 1 + \cdots 1}_{\text{endInd-startInd+1 times}} = endInd - startInd + 1$$

$$\sum_{i=1}^{n} ca_i = c\sum_{i=1}^{n} a_i$$

$$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \in \Theta(n^2)$$

$$\sum_{i=1}^{n} i^k \in \Theta(n^{k+1}) \ldots$$

Example: Matrix Multiplication

Algorithm $MatrixMultiplication(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$
//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two n-by-n matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
        $C[i,j] \leftarrow 0.0$
        for $k \leftarrow 0$ **to** $n-1$ **do**
            $C[i,j] \leftarrow C[i,j] + A[i,k] * B[k,j]$
return C

Input size? ***n*** **Basic operations?** ***Multiplication***

$$C_{worst}(n) = C_{best}(n) = C_{average}(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = n^3 \in \Theta(n^3)$$

Example: Selection sort

ALGORITHM *SelectionSort*($A[0..n-1]$)

```
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]   min ← j
    swap A[i] and A[min]
```

Input size? ***n***

Basic operations?

Comparison

$$C_{worst}(n) = C_{best}(n) = C_{average}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} (n-i-1) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Example: Find the Number of Binary Digits

Find the Number of Binary Digits in the Binary Representation of a Positive Decimal Integer

```
ALGORITHM Binary(n)
// Input: A positive decimal integer n
// Output: The number of binary digits
//          in n's binary representation
count ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

Input size? n

Basic operations? $/$

$$C_{worst}(n), C_{best}(n), C_{average}(n) = ?$$

$$\lfloor \log_2 n \rfloor$$

Example: Element Uniqueness

Check whether all the elements in a given array are distinct

- Input: An array A[0…n-1]
- Output: Return “true” if all the elements in A are distinct and “false” otherwise

ALGORITHM $UniqueElements(A[0..n-1])$

for $i \leftarrow 0$ **to** n-2 **do**

  for $j \leftarrow i+1$ **to** n-1 **do**

    if $A[i] = A[j]$ **return false**

return true

Input size? ***n***

Basic operations?

Comparison

$$C_{worst}(n), C_{best}(n) = ?$$

$$C_{best}(n) \in \Theta(1) \qquad C_{worst}(n) \in \Theta(n^2)$$