Asymptotic Analysis

CSE 373
Data Structures & Algorithms
Ruth Anderson
Autumn 2010

Today's Outline

- **Announcements**
  – Assignment #1 due Thurs, Oct 7 at 11:45pm
- **Asymptotic Analysis**

Exercise

2	3	5	16	37	50	73	75	126

```
bool ArrayFind(int array[], int n, int key){
  // Insert your algorithm here



}
```

What algorithm would you choose to implement this code snippet?

Analyzing Code

Basic Java operations	Constant time
Consecutive statements	Sum of times
Conditionals	Larger branch plus test
Loops	Sum of iterations
Function calls	Cost of function body
Recursive functions	Solve recurrence relation

Analyze your code!

Linear Search Analysis

```
bool LinearArrayFind(int array[],
                     int n,
                     int key ) {
  for( int i = 0; i < n; i++ ) {
      if( array[i] == key )
            // Found it!
            return true;
  }
  return false;
}
```

Best Case:

Worst Case:

Binary Search Analysis

```
bool BinArrayFind( int array[], int low,
                   int high, int key ) {
  // The subarray is empty
  if( low > high ) return false;

  // Search this subarray recursively
  int mid = (high + low) / 2;
  if( key == array[mid] ) {
      return true;
  } else if( key < array[mid] ) {
      return BinArrayFind( array, low,
                           mid-1, key );
  } else {
      return BinArrayFind( array, mid+1,
                           high, key );
}
```

Best case:

Worst case:

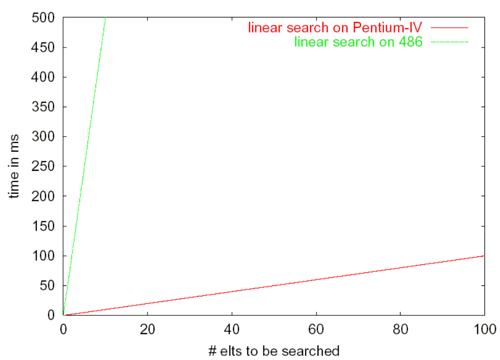
Solving Recurrence Relations

1. Determine the recurrence relation. What is the base case(s)?
2. "Expand" the original relation to find an equivalent general expression *in terms of the number of expansions*.
3. Find a closed-form expression by setting *the number of expansions* to a value which reduces the problem to a base case

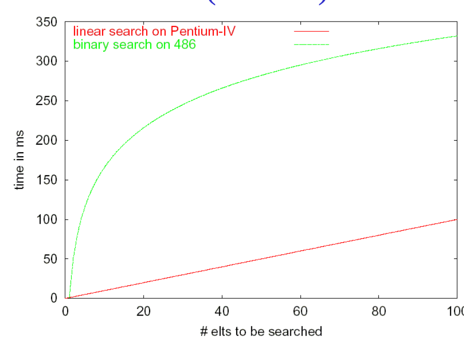
Linear Search vs Binary Search

So … which algorithm is better?
What tradeoffs can you make?

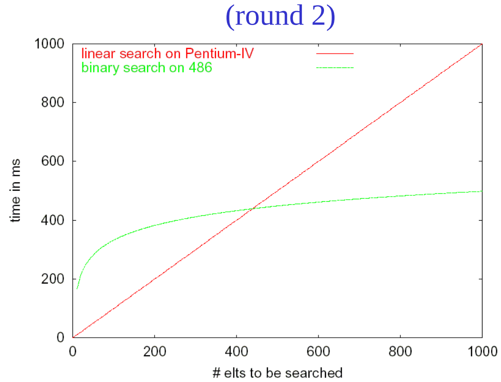
Fast Computer vs. Slow Computer



Fast Computer vs. Smart Programmer (round 1)



Fast Computer vs. Smart Programmer (round 2)



Asymptotic Analysis

- Asymptotic analysis looks at the *order* of the running time of the algorithm
  - A valuable tool when the input gets "large"
  - Ignores the *effects of different machines* or *different implementations* of the same algorithm
- Intuitively, to find the asymptotic runtime, throw away the constants and low-order terms
  - Linear search is $T(n) = 3n + 3 \in \Theta(\boldsymbol{n})$
  - Binary search is $T(n) = 5 \log_2 n + 7 \in \Theta(\mathbf{log}\ \boldsymbol{n})$

Remember: the fastest algorithm has the slowest growing function for its runtime

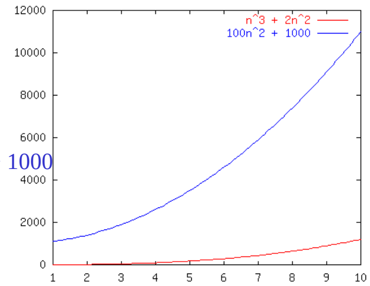
Asymptotic Analysis

- Eliminate low order terms
  - $4n + 5 \Rightarrow$
  - $0.5\, n \log n + 2n + 7 \Rightarrow$
  - $n^3 + 2^n + 3n \Rightarrow$
- Eliminate coefficients
  - $4n \Rightarrow$
  - $0.5\, n \log n \Rightarrow$
  - $n \log n^2 =>$

Order Notation: Intuition

$f(n) = n^3 + 2n^2$

$g(n) = 100n^2 + 1000$



Although not yet apparent, as n gets "sufficiently large", $f(n)$ will be "greater than or equal to" $g(n)$

Definition of Order Notation

- Upper bound: $T(n) = O(f(n))$ Big-O

  Exist constants c and n' such that

  $T(n) \leq c\, f(n)$ for all $n \geq n'$
- Lower bound: $T(n) = \Omega(g(n))$ Omega

  Exist constants c and n' such that

  $T(n) \geq c\, g(n)$ for all $n \geq n'$
- Tight bound: $T(n) = \Theta(f(n))$ Theta

  When both hold:

  $T(n) = O(f(n))$

  $T(n) = \Omega(f(n))$

Order Notation: Definition

O(f(n)) : <u>a set or class of functions</u>

$g(n) \in O(f(n))$ iff there exist consts c and n_0 such that:

$g(n) \leq c\, f(n)$ for all $n \geq n_0$

Example: $g(n) = 1000n$ vs. $f(n) = n^2$

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

Pick: n0 = 1000, c = 1

Notation Notes

Note: Sometimes, you'll see the notation:

$g(n) = O(f(n))$.

This is equivalent to:

$g(n)$ is $O(f(n))$.

However: The notation

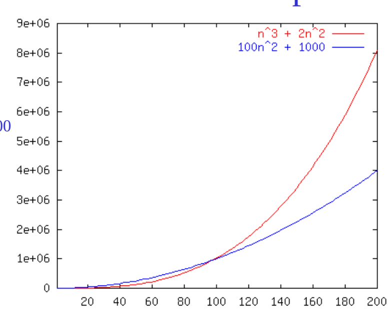
$O(f(n)) = g(n)$ is meaningless!

(in other words big-O "equality" is not symmetric)

Order Notation: Example

$f(n) = n^3 + 2n^2$

$g(n) = 100n^2 + 1000$



$100n^2 + 1000 \leq 5\,(n^3 + 2n^2)$ for all $n \geq 19$

So $g(n)$ is $O(f(n))$

Big-O: Common Names

- constant: O(1)
- logarithmic: O(log n) ($\log_k n$, $\log n^2$ is O(log n))
- log-squared: $O(\log^2 n)$ ($\log^2 n$) = (log n)(logn)
- linear: O(n)
- log-linear: O(n log n)
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- polynomial: $O(n^k)$ (k is a constant)
- exponential: $O(c^n)$ (c is a constant > 1)

Meet the Family

- O(f(n)) is the set of all functions asymptotically less than or equal to f(n)
  - o(f(n)) is the set of all functions asymptotically strictly less than f(n)
- Ω(f(n)) is the set of all functions asymptotically greater than or equal to f(n)
  - ω(f(n)) is the set of all functions asymptotically strictly greater than f(n)
- Θ(f(n)) is the set of all functions asymptotically equal to f(n)

Meet the Family, Formally

- $g(n) \in O(f(n))$ iff
  There exist c and n_0 such that $g(n) \leq c\, f(n)$ for all $n \geq n_0$
  - $g(n) \in o(f(n))$ iff
    There exists a n_0 such that $g(n) < c\, f(n)$ for all c and $n \geq n_0$

    Equivalent to: $\lim_{n\to\infty} g(n)/f(n) = 0$
- $g(n) \in \Omega(f(n))$ iff
  There exist $c>0$ and n_0 such that $g(n) \geq c\, f(n)$ for all $n \geq n_0$
  - $g(n) \in \omega(f(n))$ iff
    There exists a n_0 such that $g(n) > c\, f(n)$ for all c and $n \geq n_0$

    Equivalent to: $\lim_{n\to\infty} g(n)/f(n) = \infty$
- $g(n) \in \Theta(f(n))$ iff
  $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$

Big-Omega et al. Intuitively

Asymptotic Notation	Mathematics Relation
O	≤
Ω	≥
Θ	=
o	<
ω	>

Pros and Cons of Asymptotic Analysis

Types of Analysis

Two orthogonal axes:

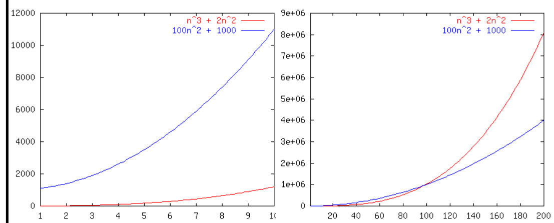
- bound flavor
  - upper bound (O, o)
  - lower bound (Ω, ω)
  - asymptotically tight (Θ)
- analysis case
  - worst case (adversary)
  - average case
  - best case
  - "amortized"

Which Function Grows Faster?

$n^3 + 2n^2$ vs. $100n^2 + 1000$

Which Function Grows Faster?

$n^3 + 2n^2$ vs. $100n^2 + 1000$

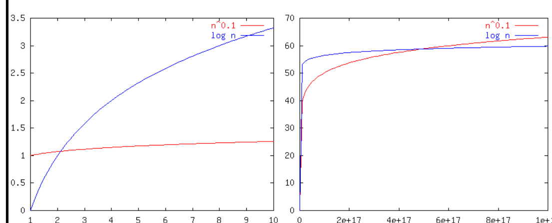


Which Function Grows Faster?

$n^{0.1}$ vs. $\log n$

Which Function Grows Faster?

$n^{0.1}$ vs. $\log n$

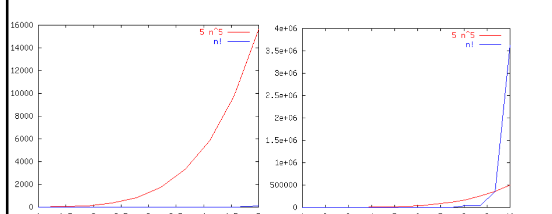


Which Function Grows Faster?

$5n^5$ vs. $n!$

Which Function Grows Faster?

$5n^5$ vs. $n!$



Nested Loops

```
for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1

for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1
```

Nested Loops

```
for i = 1 to n do
  for j = 1 to n do
    if (cond) {
        do_stuff(sum)
    } else {
        for k  = 1 to n*n
            sum += 1
```

$16n^3 \log_8(10n^2) + 100n^2 = O(n^3 \log(n))$

- Eliminate low order terms
- Eliminate constant coefficients

$16n^3 \log_8(10n^2) + 100n^2 = O(n^3 \log(n))$

- Eliminate low order terms
- Eliminate constant coefficients

$$
\begin{aligned}
&16n^3 \log_8(10n^2) + 100n^2 \\
&\Rightarrow 16n^3 \log_8(10n^2) \\
&\Rightarrow n^3 \log_8(10n^2) \\
&\Rightarrow n^3 \left[\log_8(10) + \log_8(n^2)\right] \\
&\Rightarrow n^3 \log_8(10) + n^3 \log_8(n^2) \\
&\Rightarrow n^3 \log_8(n^2) \\
&\Rightarrow n^3 2 \log_8(n) \\
&\Rightarrow n^3 \log_8(n) \\
&\Rightarrow n^3 \log_8(2) \log(n) \\
&\Rightarrow n^3 \log(n)
\end{aligned}
$$