

CSE 326 Data Structures

CSE 326 : Dave Bacon

Asymptotic Analysis

Logistics

- Project 1 – Reverse a sound file
 - Due Wed January 10, 2007 at electronically at midnight
- Homework 1 now online
 - Due Fri January 12, 2007 at beginning of lecture
- Reading (assume you finished Chapter 1)
 - Chapter 3 – (Project #1) Lists, Stacks, & Queues
 - Lists (pp. 57-81, heavy on Java, much of this should be review)
 - Stacks (pp. 82-83)
 - Applications of Stacks (pp. 83-91)
 - Queues (pp. 91-95)
 - Chapter 2 – (Topic for today, Monday) Algorithm Analysis (pp. 29-50)

Analysis of Algorithms

- Efficiency measure
 - How long the program takes
(time complexity) ← our focus today
 - How much memory the program takes
(memory complexity)
- Why analyze at all?

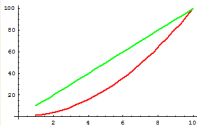
Asymptotic Analysis

- Running time as a function of the input size n
 - $T(n) = 4n + 5 + n^2$
 - $T(n) = \log_2 n + 5$
 - $T(n) = 2^n + 6n^6 - n$
- What happens as n grows?

Why Asymptotic Analysis?

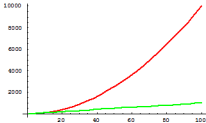
- For small n most algorithms are fast
 - Time differences too small to notice
 - External things dominate (OS, disk I/O)
- n is often LARGE in practice
 - databases, internet, graphics, etc.
- Time differences really show up for large n

Example



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

$$g(n) = 10n$$



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

$$g(n) = 10n$$

Types of Asymptotic Analysis

- Worst case
 - Guarantee running time
- Average case
 - What do we mean by average?
 - Distribution over inputs?

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

$$3n+3 = O(n)$$

Linear Search Analysis

$$n=0$$

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

array[0]
== key

$$1 + 3n + 1 + 1$$

worst \rightarrow not in array

Best Case: $4, 3$
Worst Case: $4n, 3n + 2$

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 );
    }
}
```

$4 \log_2 n + 5$

Best case:

4

Worst case:

?

$\log_2 n + \pi$

↑↑

e

$4 + T(\frac{n}{2})$

Solving Recurrence Relations

1. Determine the recurrence relation. What is the base case(s)?

$$T(n) = 4 + T\left(\frac{n}{2}\right) \leftarrow T(1) = 5$$

2. "Expand" the original relation to find an equivalent general expression *in terms of the number of expansions*.

$$T(n) = 4 + (4 + T\left(\frac{n}{4}\right))$$

3. Find a closed-form expression by setting the *number of expansions* to a value which reduces the problem to a base case

↓ K times

$$T(n) = 4K + T\left(\frac{n}{2^K}\right)$$

$$T(n) = 4 \log_2 n + 5 \quad K = \log_2 n$$

$$T\left(\frac{n}{2}\right) = 4 + T\left(\frac{n}{4}\right)$$

2, 4, 8

$$T(n) = 4 + (4 + (4 + T\left(\frac{n}{8}\right)))$$

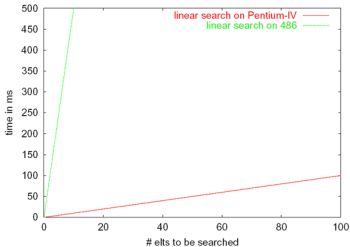
$$\frac{n}{2^K} = 1 \\ n = 2^K$$

Linear Search vs Binary Search

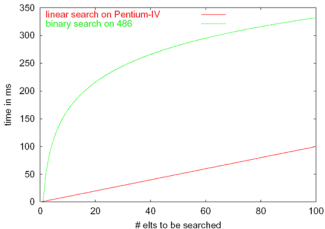
	Linear Search	Binary Search
Best Case	4	4
Worst Case	$3n+3$	$4 \log n + 5$

*So ... which algorithm is better?
What tradeoffs can you make?*

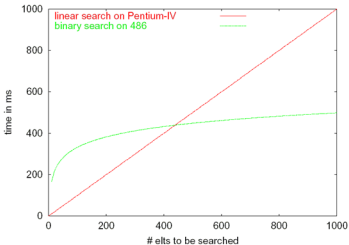
Fast Computer vs. Slow Computer



Fast Computer vs. Smart Programmer (round 1)



Fast Computer vs. Smart Programmer (round 2)



Asymptotic Analysis

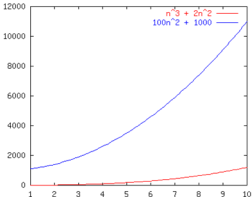
- Asymptotic analysis looks at the *order* of the running time of the algorithm
 - A valuable tool when the input gets “large”
 - Ignores the *effects of different machines* or *different implementations* of the same algorithm
- Intuitively, to find the asymptotic runtime, throw away the constants and low-order terms
 - Linear search is $T(n) = 3n + 2 \in \mathbf{O}(n)$
 - Binary search is $T(n) = 4 \log_2 n + 4 \in \mathbf{O}(\log n)$

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

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” $g(n)$

Order Notation: Definition

$O(f(n))$: a set or class of functions

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

$$g(n) \leq c f(n) \text{ for all } n \geq n_0$$

Example:

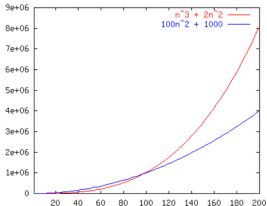
$$100n^2 + 1000 \leq 5(n^3 + 2n^2) \text{ for all } n \geq 19$$

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

Sometimes, you'll see the notation $g(n) = O(f(n))$. This is equivalent to $g(n) \in O(f(n))$.

Remember: notation $O(f(n)) = g(n)$ is meaningless!

Order Notation: Example



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

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

Big-O: Common Names



$O(1)$

$O(\log_2 n)$

$O(n)$

$O(n \log_2 n)$

$O(n^2)$

$O(n^3)$

$O(n^k)$ (k is a constant)

$O(c^n)$ (c is a constant > 1)

Log?

$$\log_k n \in O(\log_2 n)?$$

$$\log_2 n^2 \in O(\log_2 n)?$$

The Limit Method

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

Style

$$O(6n^6+5+n^{2.5})$$

$$O(2^n+5n(n-1))$$

Pros and Cons of Asymptotic Analysis

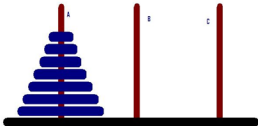
Definition of Order Notation

- Upper bound: $T(n) = O(f(n))$ Big-O
Exist constants c and n' such that
$$T(n) \leq c f(n) \text{ for all } n \geq n'$$
- Lower bound: $T(n) = \Omega(g(n))$ Omega
Exist constants c and n' such that
$$T(n) \geq c g(n) \text{ for all } n \geq n'$$
- Tight bound: $T(n) = \theta(f(n))$ Theta

When both hold:

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

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



ALGORITHM TH (n, A,B,C)

1. if $n \leq 0$ then return
 2. TH (n-1, A,C,B)
 3. A ==> B
 4. TH (n-1, C,B,A)
- end**

Logistics

- Project 1 – Reverse a sound file
 - Due Wed January 10, 2007 at electronically at midnight
- Homework 1 now online
 - Due Fri January 12, 2007 at beginning of lecture
- Reading (assume you finished Chapter 1)
 - Chapter 3 – (Project #1) Lists, Stacks, & Queues
 - Lists (pp. 57-81, heavy on Java, much of this should be review)
 - Stacks (pp. 82-83)
 - Applications of Stacks (pp. 83-91)
 - Queues (pp. 91-95)
 - Chapter 2 – (Topic for today, Monday) Algorithm Analysis (pp. 29-50)

Office Hours

- Dave Bacon, Tu 4:00-5:00, CSE 460
- Ruth Anderson, M 3:30-4:30, CSE 360

- Ethan Phelps-Goodman, Th 10:30-11:30, CSE 218
- Jonah Cohen, W 1:30-2:30, TBA

- David Wu, W 4:00-5:00 (in lab CSE 002/003), Th 3:30-4:30 (in CSE 218)