CSE 326: Data Structures
Topic 2: Asymptotic Analysis

Luke McDowell
Summer Quarter 2003

Course Policies – Updated

- Written homeworks
  - Due at the start of class on due date
  - No late homeworks accepted
- Programming homeworks
  - Turned in electronically before 11pm on due date
  - Once per quarter: use your "late day" for extra 24 hours – **Must email TA**
- Work in teams only on explicit team projects
  - Appropriate *discussions* encouraged – see website
- Approximate Grading
  - Weekly assignments: 35%
  - Midterm: 20% Friday July 25, in class
  - Final: 30% Friday Aug. 22 in class
  - **Best of above 3:** 10%
  - Participation: 5%

Analysis of Algorithms

- Efficiency measure
  - how long the program runs — time complexity
  - how much memory it uses — space complexity
    - For today, we'll focus on time complexity only
- *Why analyze at all?*
  - Confidence: algorithm will work well in practice
  - Insight : alternative, better algorithms

Asymptotic Analysis

- Complexity as a function of input size n

  $T(n) = 4n + 5$

  $T(n) = 0.5 n \log n - 2n + 7$

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

- *What happens as n grows?*

Why do we care?

- Most algorithms are fast for small n
  - Time difference too small to be noticeable
  - External things dominate (OS, disk I/O, …)
- BUT n is often large in practice
  - Databases, internet, graphics, …
- Time difference really shows up as n grows!

Luke Takes a Break

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?

Luke Takes a Break: Simplifying assumptions

- Ideal single-processor machine (serialized operations)
- "Standard" instruction set (load, add, store, etc.)
- All operations take 1 time unit (including, for our purposes, each Java or C++ statement

LTaB: Analyzing Code

Basic Java/C++ 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

LTaB: Linear Search Analysis

```
bool ArrayFind(   int array[],
                  int n,
                  int key )
{

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

}
```

- Best Case:
- Worst Case:

LTaB: Binary Search Analysis

```
bool ArrayFind( int array[], int s,
                int e, int key ) {
  // The subarray is empty
  if( s > e )
      return false;

  // Search this subarray
  int mid = (e + s) / 2;
  if( array[key] == array[mid] ) {
      return true;
  } else if( key < array[mid] ) {
      return ArrayFind( array, s,
                        mid-1, key );
  } else {
      return ArrayFind( array, mid+1,
                        e, key );
}
```

- Best case:
- Worst case:

Back to work: Solving Recurrence Relations

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

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

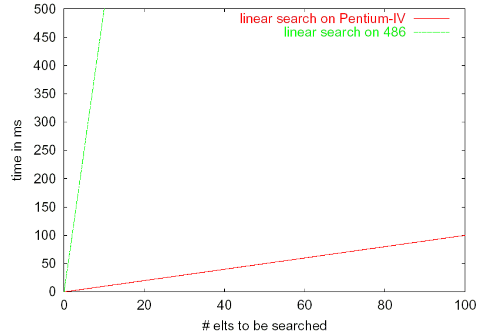
3. Find a closed-form expression by setting *the number of expansions* to a value which reduces the problem to a base case

Linear Search vs Binary Search

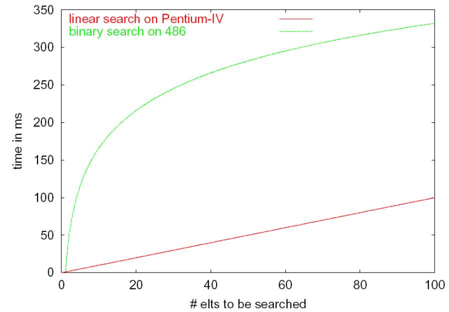
	Linear Search	Binary Search
Best Case		
Worst Case		

So … which algorithm is best?
What tradeoffs did you make?

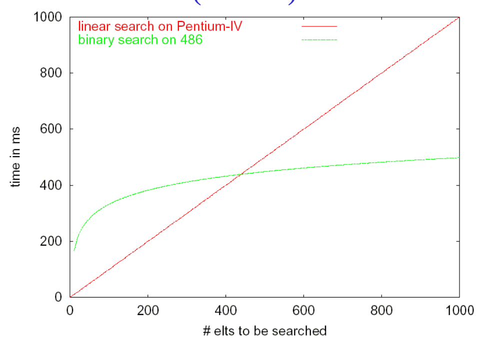
Fast Computer vs. Slow Computer



Fast Computer vs. Smart Programmer (round 1)



Fast Computer vs. Smart Programmer (round 2)



Asymptotic Analysis

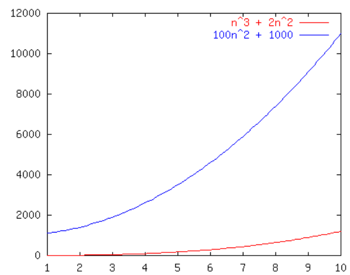
- Asymptotic analysis looks at the *order* of the running time of the algorithm
  - A valuable tool when the input gets "large"
  - Ignores the *effects of different machines* or *different implementations* of the same algorithm
- Intuitively, to find the asymptotic runtime, throw away the constants and low-order terms
  - Linear search is T(n) = 2n + 1 ? O(n)
  - Binary search is T(n) = 4 $\log_2 n$ + 2 ? 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

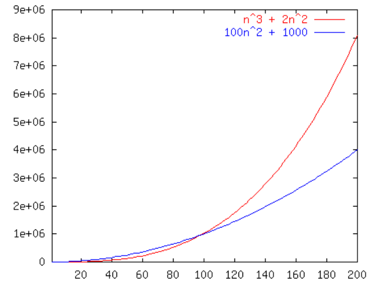
O(f(n)) is a *set of functions*

g(n) ? O(f(n)) iff
There exist c and n_0 such that g(n) ? c f(n) for all n ? n_0

Example:
$100n^2 + 1000$? 5 ($n^3 + 2n^2$) for all n ? 19
So g(n) ? O(f(n))

Sometimes, you'll see the notation g(n) = O(f(n)). This equivalent to g(n) ? O(f(n)). However, the notation O(f(n)) = g(n) is *not* correct

Order Notation: Example

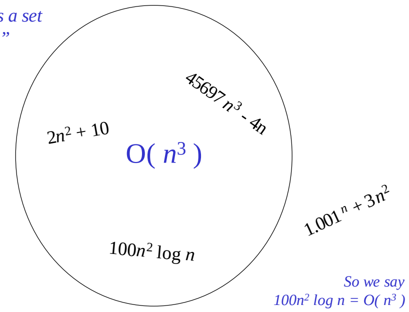


$100n^2 + 1000$? 5 $(n^3 + 2n^2)$ for all n ? 19

So g(n) ? O(f(n))

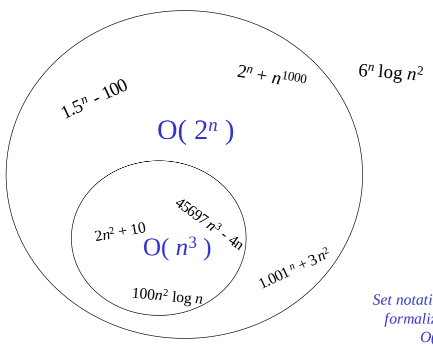
Oops: Set Notation

"O(f(n)) is a set of functions"



So we say both
$100n^2 \log n$ = O(n^3) and
$100n^2 \log n$? O(n^3)

Set Notation



Set notation allows us to formalize our intuition
O(n^3) ? O(2^n)

Big-O Common Names

- constant: O(1)
- logarithmic: O(log n) ($\log_k n$, $\log n^2$? O(log n))
- poly-log: O($\log^k n$)
- linear: O(n)
- log-linear: O(n log n)
- superlinear: O(n^{1+c}) (c is a constant > 0)
- 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)
- ? (f(n)) is the set of all functions asymptotically greater than or equal to f(n)
  - ? (f(n)) is the set of all functions asymptotically strictly greater than f(n)
- ?(f(n)) is the set of all functions asymptotically equal to f(n)

Meet the Family Formally (don't worry about dressing up)

- g(n) ? O(f(n)) iff
  There exist c and n_0 such that g(n) ? c f(n) for all n ? n_0
  - g(n) ? o(f(n)) iff
    There exists a n_0 such that g(n) < c f(n) for all c and n ? n_0
- g(n) ? ? (f(n)) iff
  There exist c and n_0 such that g(n) ? c f(n) for all n ? n_0
  - g(n) ? ? (f(n)) iff
    There exists a n_0 such that g(n) > c f(n) for all c and n ? n_0
- g(n) ? ?(f(n)) iff
  g(n) ? O(f(n)) and g(n) ? ? (f(n))

Big-Omega et al. Intuitively

Asymptotic Notation	Mathematics Relation
O	?
?	?
?	=
o	<
?	>

True or False?

$10{,}000\, n^2 + 25n$? ?(n^2)	
$10^{-10}\, n^2$? ?(n^2)	
$n^3 + 4$? ?(n^3)	
$n \log n$? O(2^n)	
$n \log n$? ? (n^2)	
$n^3 + 4$? o(n^4)	

Types of Analysis

Two orthogonal axes:

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

LTaB: Pros and Cons of Asymptotic Analysis

Proof by...

- Counterexample
  - show an example which does not fit with the theorem
  - QED (the theorem is disproven)
- Contradiction
  - assume the opposite of the theorem
  - derive a contradiction
  - QED (the theorem is proven)
- Induction
  - prove for a base case (e.g., n = 1)
  - assume for an anonymous value (n)
  - prove for the next value (n + 1)
  - QED

Inductive Proof of Correctness

```
int sum(int v[], int n)
{
  if (n==0) return 0;
  else return v[n-1]+sum(v,n-1);
}
```

Theorem: sum(v,n) correctly returns sum of 1st n elements of array v for any n.

Basis Step: Program is correct for n=0; returns 0.

Inductive Hypothesis (n=k): Assume sum(v,k) returns sum of first k elements of v.

Inductive Step (n=k+1): sum(v,k+1) returns v[k]+sum(v,k), which is the same of the first k+1 elements of v.

Inductive Proof (Binary Search)

If you know the closed form solution,
you can validate it by ordinary induction

$T(1)\ ?\ b\ ?\ c\log 1\ ?\ b$ base case

Assume $T(n)\ ?\ b\ ?\ c\log n$ hypothesis

$T(2n)\ ?\ T(n)\ ?\ c$ definition of T(n)

$T(2n)\ ?\ (b\ ?\ c\log n)\ ?\ c$ by induction hypothesis

$T(2n)\ ?\ b\ ?\ c((\log n)\ ?\ 1)$

$T(2n)\ ?\ b\ ?\ c((\log n)\ ?\ (\log 2))$

$T(2n)\ ?\ b\ ?\ c\log(2n)$ Q.E.D.

Thus: $T(n)\ ?\ ?\ (\log n)$

Asymptotic Analysis Summary

- Determine what characterizes a problem's size
- Express how much resources (time, memory, etc.) an algorithm requires as a function of input size using O(•), ? (•), ?(•)
  - worst case
  - best case
  - average case
  - common case
  - overall

To Do

- Continue Homework 1
  - Due Monday, June 30 at 11 PM sharp!
  - Bring questions to section tomorrow
- Sign up for 326 mailing list(s)
- Continue reading 1.1-1.3, Chapters 2 and 3 in the book
  - Also start/skim on next sections: 4.1 (introduction to trees), and sections 6.1-6.4 (priority queues and binary heaps)