CSE 332: Data Structures

Asymptotic Analysis II

Richard Anderson, Steve Seitz
Winter 2014

Announcements

- Due next week
  - Project 1A, Monday, 11:59 PM
  - Homework 1, Wednesday, beginning of class
  - Project 1B, Thursday, 11:59 PM

Linear Search Analysis

```
bool LinearArrayContains(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:
  4

Worst Case:
  3n+3

Binary Search Analysis

2	3	5	16	37	50	73	75

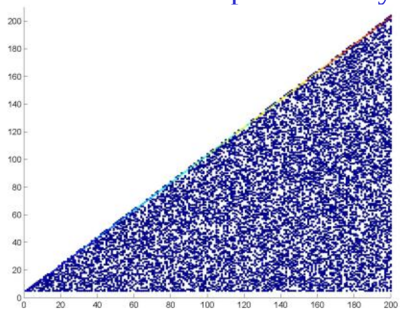
```
bool BinArrayContains( int array[], int low, int high, int key ) {
  // The subarray is empty
  if( low > high ) return false;

  // 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:
  5 at [middle]

Worst case:
  $7 \lfloor \log n \rfloor + 9$

Linear search—empirical analysis

time (# ops)

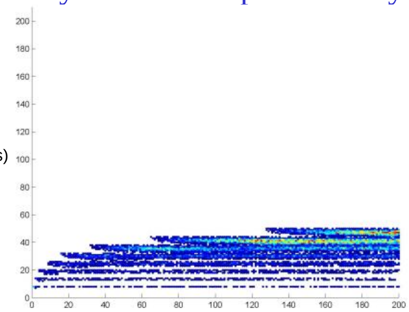


N (= array size)

Each search produces a dot in above graph.
Blue = less frequently occurring, Red = more frequent

Binary search—empirical analysis

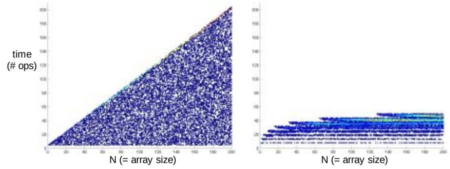
time (# ops)



N (= array size)

Each search produces a dot in above graph.
Blue = less frequently occurring, Red = more frequent

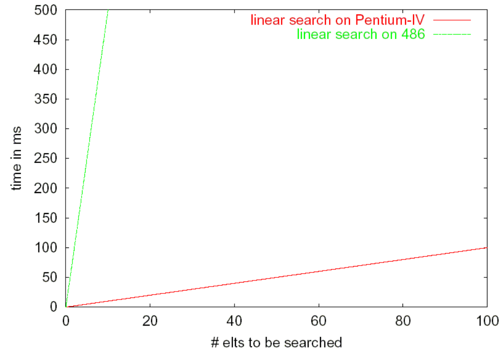
Empirical comparison



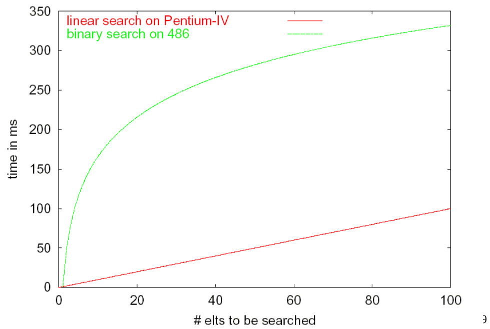
Linear search Binary search

Gives additional information

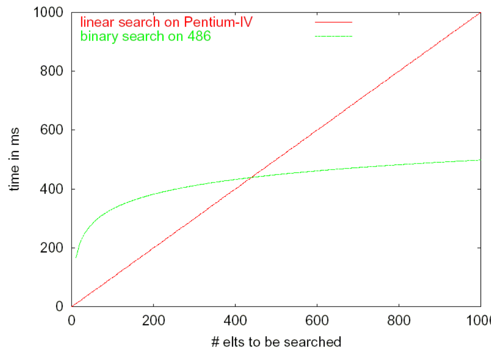
Fast Computer vs. Slow Computer



Fast Computer vs. Smart Programmer (small data)



Fast Computer vs. Smart Programmer (big data)



Asymptotic Analysis

- Consider only the *order* of the running time
  - A valuable tool when the input gets "large"
  - **Ignores** the effects of ***different machines*** or ***different implementations*** of same algorithm

Asymptotic Analysis

- To find the asymptotic runtime, throw away the constants and low-order terms
  - Linear search is $T^{LS}_{worst}(n) = 3n + 3 \in O(n)$
  - Binary search is $T^{BS}_{worst}(n) = 7\lfloor \log_2 n \rfloor + 9 \in O(\log n)$

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

Asymptotic Analysis

Eliminate low order terms

- $4n + 5 \Rightarrow$
- $0.5 \text{ n log n} + 2n + 7 \Rightarrow$
- $n^3 + 3\ 2^n + 8n \Rightarrow$

Eliminate coefficients

- $4n \Rightarrow$
- $0.5 \text{ n log n} \Rightarrow$
- $3\ 2^n$ =>

Properties of Logs

Basic:

- $A^{\log_A B} = B$
- $\log_A A =$

Independent of base:

- $\log(AB) =$
- $\log(A/B) =$
- $\log(A^B) =$
- $\log((A^B)^C) =$

Properties of Logs

Changing base $\rightarrow$ multiply by constant

- For example: $\log_2 x = 3.22 \log_{10} x$
- More generally

$$\log_A n = \left(\frac{1}{\log_B A}\right) \log_B n$$

- Means we can ignore the base for asymptotic analysis (since we're ignoring constant multipliers)

Another example

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

- Eliminate low-order terms
- Eliminate constant coefficients

Comparing functions

- f(n) is an **upper bound** for h(n) if h(n) ≤ f(n) for all n

This is too strict – we mostly care about *large* n

Still too strict if we want to ignore *scale factors*

Definition of Order Notation

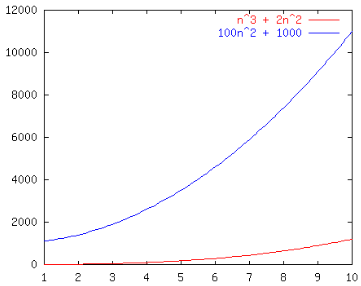
- h(n) ϵ O(f(n)) Big-O "Order"
  if there exist positive constants c and n_0 such that $h(n) \leq c\, f(n)$ for all $n \geq n_0$

O(f(n)) defines a class (set) of functions

Order Notation: Intuition

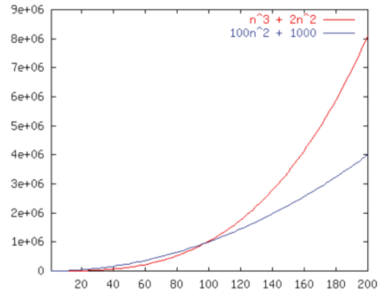
$a(n) = n^3 + 2n^2$

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



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

Order Notation: Example



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

So $100n^2 + 1000 \in O(n^3 + 2n^2)$

Example

$h(n) \in O(\ f(n)\ )$ iff there exist positive constants c and n_0 such that:
$h(n) \leq c\ f(n)$ for all $n \geq n_0$

Example:

$100n^2 + 1000 \leq 1\ (n^3 + 2n^2)$ for all $n \geq 100$

So $100n^2 + 1000 \in O(n^3 + 2n^2\ )$

Constants are not unique

$h(n) \in O(\ f(n)\ )$ iff there exist positive constants c and n_0 such that:
$h(n) \leq c\ f(n)$ for all $n \geq n_0$

Example:

$100n^2 + 1000 \leq 1\ (n^3 + 2n^2)$ for all $n \geq 100$

$100n^2 + 1000 \leq 1/2\ (n^3 + 2n^2)$ for all $n \geq 198$

Another Example: Binary Search

$h(n) \in O(\ f(n)\ )$ iff there exist positive constants c and n_0 such that:
$h(n) \leq c\ f(n)$ for all $n \geq n_0$

Is $7\log_2 n + 9 \in O\ (\log_2 n)$?

Order Notation: Worst Case Binary Search

Some Notes on Notation

Sometimes you'll see (e.g., in Weiss)

$$h(n) = O(\ f(n)\ )$$

or

$$h(n) \text{ is } O(\ f(n)\ )$$

These are equivalent to

$$h(n) \in O(\ f(n)\ )$$

Big-O: Common Names

- constant: $O(1)$
- logarithmic: $O(\log n)$ ($\log_k n$, $\log n^2 \in O(\log n)$)
- 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)

Asymptotic Lower Bounds

- $\Omega(\ g(n)\ )$ is the set of all functions asymptotically greater than or equal to $g(n)$
- $h(n) \in \Omega(\ g(n)\ )$ iff
  There exist $c>0$ and $n_0>0$ such that $h(n) \geq c\, g(n)$ for all $n \geq n_0$

Asymptotic Tight Bound

- $\theta(\ f(n)\ )$ is the set of all functions asymptotically equal to $f(n)$
- $h(n) \in \theta(\ f(n)\ )$ iff
  $h(n) \in O(\ f(n)\ )$ and $h(n) \in \Omega(f(n)\ )$
  - This is equivalent to: $\lim_{n\to\infty} h(n)/f(n) = c \neq 0$

Full Set of Asymptotic Bounds

- $O(\ f(n)\ )$ 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 $f(n)$
- $\Omega(\ g(n)\ )$ is the set of all functions asymptotically greater than or equal to $g(n)$
  - $\omega(\ g(n)\ )$ is the set of all functions asymptotically strictly greater than $g(n)$
- $\theta(\ f(n)\ )$ is the set of all functions asymptotically equal to $f(n)$

Formal Definitions

- $h(n) \in O(\ f(n)\ )$ iff
  There exist $c>0$ and $n_0>0$ such that $h(n) \leq c\, f(n)$ for all $n \geq n_0$
- $h(n) \in o(f(n))$ iff
  There exists an $n_0>0$ such that $h(n) < c\, f(n)$ for all $c>0$ and $n \geq n_0$
  - This is equivalent to: $\lim_{n\to\infty} h(n)/f(n) = 0$
- $h(n) \in \Omega(\ g(n)\ )$ iff
  There exist $c>0$ and $n_0>0$ such that $h(n) \geq c\, g(n)$ for all $n \geq n_0$
- $h(n) \in \omega(\ g(n)\ )$ iff
  There exists an $n_0>0$ such that $h(n) > c\, g(n)$ for all $c>0$ and $n \geq n_0$
  - This is equivalent to: $\lim_{n\to\infty} h(n)/g(n) = \infty$
- $h(n) \in \theta(\ f(n)\ )$ iff
  $h(n) \in O(\ f(n)\ )$ and $h(n) \in \Omega(f(n)\ )$
  - This is equivalent to: $\lim_{n\to\infty} h(n)/f(n) = c \neq 0$

Big-Omega et al. Intuitively

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

Complexity cases (revisited)

Problem size **N**

- **Worst-case complexity**: **max** # steps algorithm takes on "most challenging" input of size **N**
- **Best-case complexity: min** # steps algorithm takes on "easiest" input of size **N**
- **Average-case complexity**: **avg** # steps algorithm takes on *random* inputs of size **N**
- **Amortized complexity**: **max** total # steps algorithm takes on **M** "most challenging" *consecutive* inputs of size **N**, divided by **M** (i.e., divide the max total by **M**).

Bounds vs. Cases

Two orthogonal axes:

- Bound Flavor
  - Upper bound (O, o)
  - Lower bound (Ω, ω)
  - Asymptotically tight (θ)
- Analysis Case
  - Worst Case (Adversary), $T_{worst}(n)$
  - Average Case, $T_{avg}(n)$
  - Best Case, $T_{best}(n)$
  - Amortized, $T_{amort}(n)$

One can estimate the bounds for any given case.

Bounds vs. Cases

Pros and Cons of Asymptotic Analysis

Big-Oh Caveats

- Asymptotic complexity (Big-Oh) considers only ***large n***
  - You can "abuse" it to be misled about trade-offs
  - Example: $n^{1/10}$ vs. `log` n
    - Asymptotically $n^{1/10}$ grows more quickly
    - But the "cross-over" point is around 5 * 10^{17}
    - So $n^{1/10}$ better for almost any real problem
- Comparing O() for ***small n*** values can be misleading
  - Quicksort: O(nlogn)
  - Insertion Sort: O(n^2)
  - Yet in reality Insertion Sort is faster for small n
  - We'll learn about these sorts later