CSE 332: Data Structures and Parallelism

Fall 2022

Richard Anderson

Lecture 3: Algorithm Analysis

Announcements

- Project #1: Released
  - Due Thursday, Oct 13
- Exercise #1: Due tonight, 11:59 pm
- Reading: Weiss, for Monday and Wednesday
  - Priority Queues, 6.1-6.5

Analyzing Performance

- Focus on Worst Case Time Complexity
  - **max** # steps algorithm takes on input of size **N**
  - Run time is a function of **N**

Basic operations	Constant time
Consecutive statements	Sum of times
Conditionals	Test, plus larger branch cost
Loops	Sum of iterations
Function calls	Cost of function body
Recursive functions	Solve recurrence relation…

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:

Worst case:

Solving Recurrences
$T(n) = T(n/2) + 7;\ \ T(1) = 9$

1. Determine the recurrence relations and base cases

2. Expand relation in terms of number of expansions k

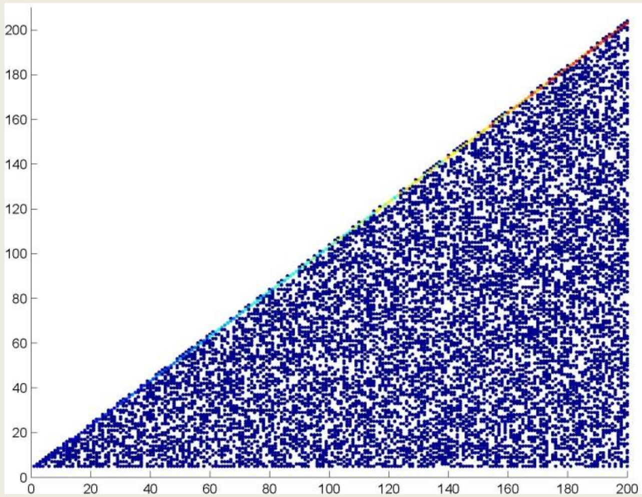
3. Find a closed form by setting k to value that reduces problem to the base case

Linear Search vs Binary Search

	Linear Search	Binary Search
Best Case	4	5 at [middle]
Worst Case	3n+3	$7 \lfloor \log n \rfloor + 9$

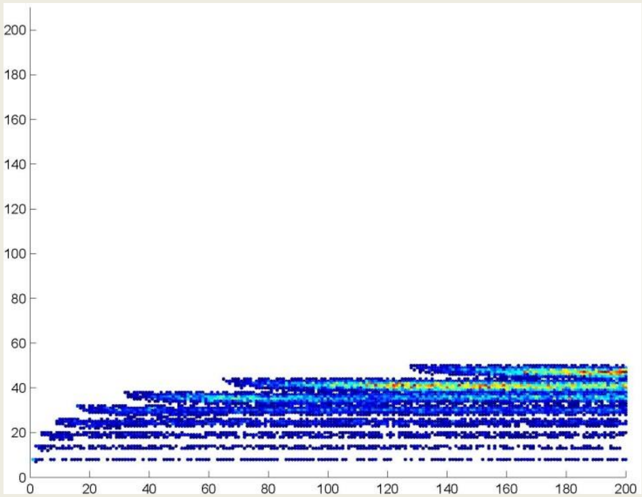
Empirical comparison

time
(# ops)



N (= array size)

Linear search



N (= array size)

Binary search

Asymptotic Analysis

- Consider only the *order of growth* of the running time
  - Runtime a function of input size

  - 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 $T_{worst}^{LS}(n) = 3n + 3 \in O(n)$

  – Binary search is $T_{worst}^{BS}(n) = 7\lfloor \log_2 n \rfloor + 9 \in O(\log n)$

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

Asymptotic Analysis

Eliminate low order and coefficients

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

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

- Eliminate low-order terms

- Eliminate constant coefficients

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

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

- $h(n) \in O(f(n))$ Big-O "Order"

if there exist positive constants c and n_0

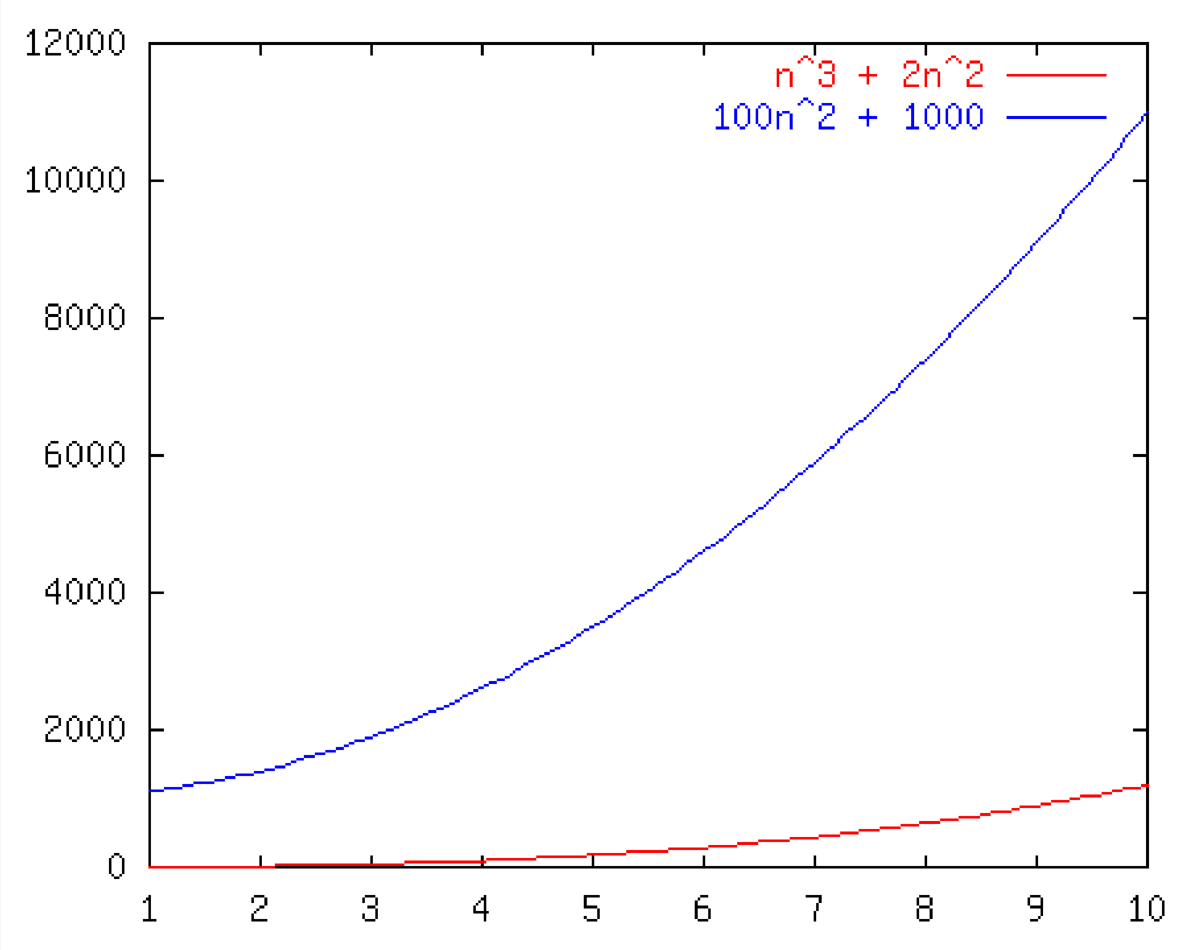
such that $h(n) \le c\, f(n)$ for all $n \ge n_0$

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

Order Notation: Intuition

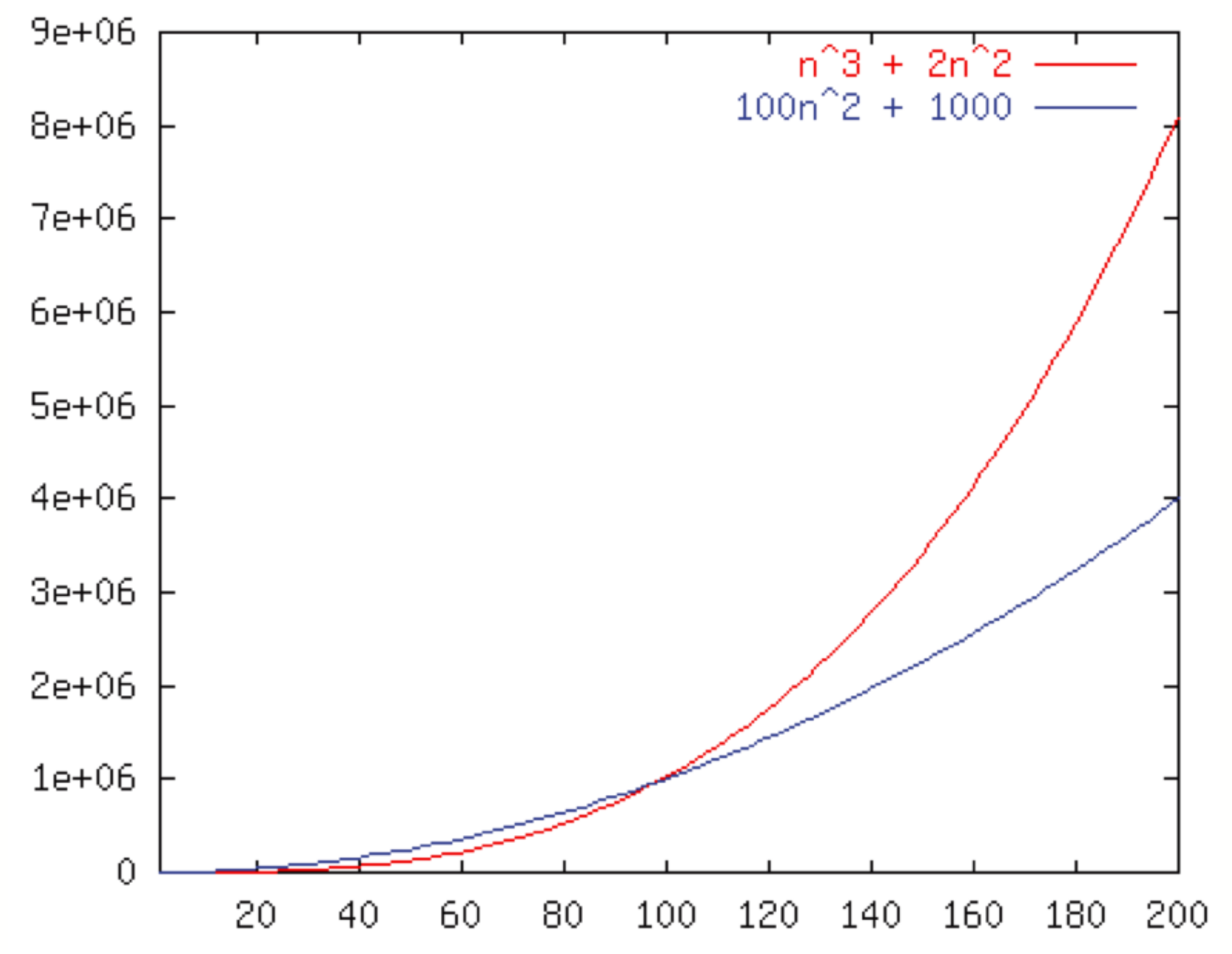
$a(n) = n^3 + 2n^2$

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



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

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 $f(n)$

- Ω($g(n)$) is the set of all functions asymptotically greater than or equal to $g(n)$
  - ω($g(n)$) is the set of all functions asymptotically strictly greater than $g(n)$

- θ($f(n)$) 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	$\leq$
Ω	$\geq$
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.