CSE 373: Asymptotic Analysis

Chapter 2

Recall: `FindMajor()`

```
/* print students in a class with major */
void FindMajor(class c,dept major) {
  int i;

  for (i=0;i<num_students;i++) {
    if (c[i].major == major) {
      printf("%s %s\n",c[i].first,c[i].last);
    }
  }
}
```

How fast is this routine?

Exact Times are Tricky

How much time does `FindMajor()` require?

– number of iterations × work per iteration:

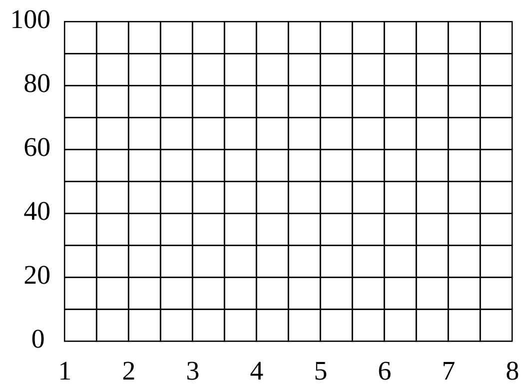
Simplifying Assumption

Constants are insignificant compared to the *asymptotic* behavior of the program

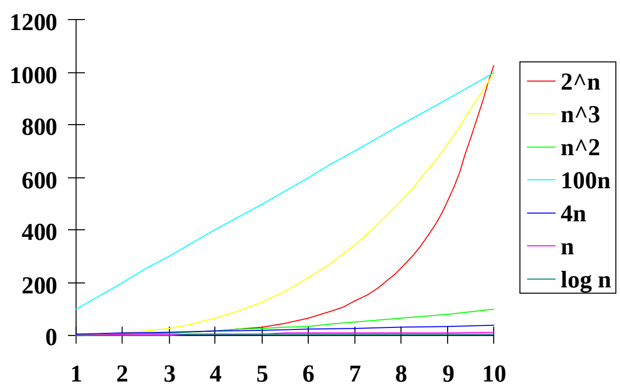
– expressed as a function of the problem size

– expressed using functions like: n, n^2, $\log n$, 2^n, etc.

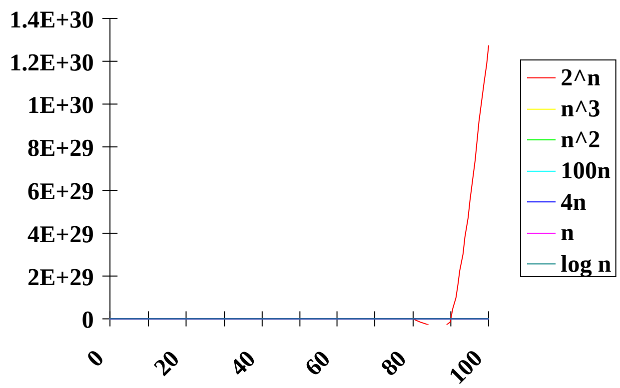
Getting some Intuition...



Using the Computer...

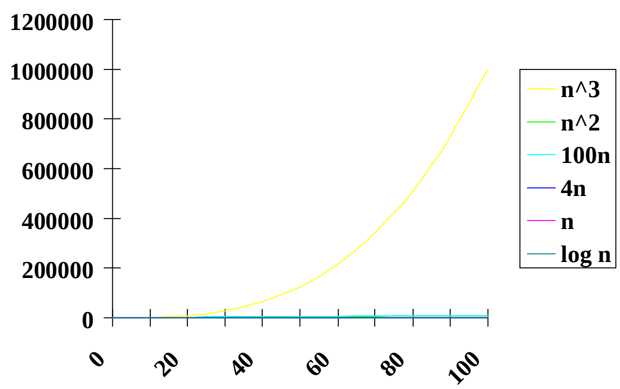


On A Larger Scale...



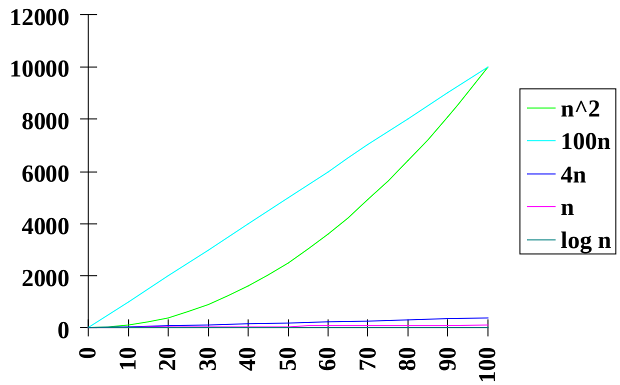
UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Ignoring 2^n



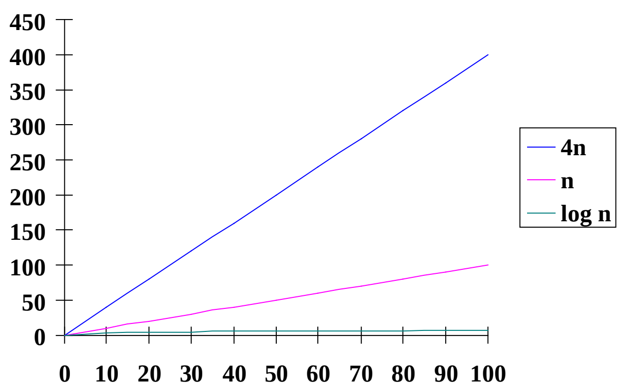
UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Ignoring n^3



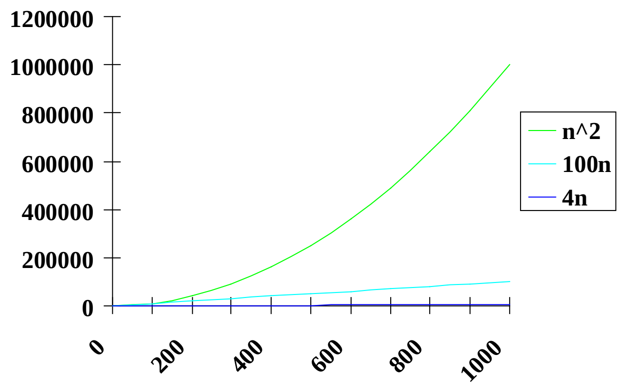
UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Smallest Functions Only



UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

On Yet a Larger Scale



The Moral

Performance can be broken down into *primary* and *secondary effects*

- *primary effects*: asymptotic growth pattern
- *secondary effects*: constant factors, less significant terms

- In this class, we'll mainly be concerned with primary effects (*asymptotic analysis*)
- In the real world, secondary effects are also often worth paying attention to (*after* the primary ones)

Formally…

Given an algorithm whose running time is $T(n)$…

- $T(n) = O(f(n))$ if there are positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$
  - $\log n, n, 100n = O(n)$
- $T(n) = \Omega(f(n))$ if there are positive constants c and n_0 such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$
  - $n, n^2, 100 \cdot 2^n, n^3 \log n = \Omega(n)$
- $T(n) = \Theta(f(n))$ if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$
  - $n, 2n, 100n, n + \log n = \Theta(n)$

Typical Growth Rates (in order)

constant:	$O(1)$
logarithmic:	$O(\log n)$
log-squared:	$O(\log^2 n)$
linear:	$O(n)$
"n log n":	$O(n \cdot \log n)$
quadratic:	$O(n^2)$
cubic;	$O(n^3)$
exponential:	$O(2^n)$

General Rules of Thumb

- Constant factors can always be dropped
  - $5n = O(n)$
- In sums, smaller terms can always be dropped
  - $3n \cdot \log n + n^2 + \log n = O(n^2)$

Analyzing Code

- *C/C++ ops* – constant value
- *consecutive statements* – add individual costs
- *loops* – multiply cost of loop body by number of iterations
- *conditionals* – maximum cost of branches
- *function calls* – evaluate cost of function body

Above all, use your brain

Reconsider: `FindMajor()`

```
/* print students in a class with major */
void FindMajor(class c,dept major) {
  int i;
  for (i=0;i<num_students;i++) {
    if (c[i].major == major) {
      cout << c[i].first << c[i].last;
    }
  }
}
```

How fast is this routine? At best?

At worst?

On average?

`FindAMajor()`

```
/* return pointer to a student in major */
student *FindAMajor(class c,dept major) {
  int i;
  for (i=0;i<num_students;i++) {
    if (c[i].major == major) {
      return &(c[i]);
    }
  }
}
```

What's the best case for this routine?

The worst case?

The average case?

Asymptotic Analysis

- Determine what characterizes a problem's size
- Express how much time and memory an algorithm requires as a function of its problem size using O(), Ω(), or Θ()
  - worst case
  - best case
  - average case
  - common case
  - overall

Examples from Lecture

	Prob Size	*Space*	*Best Time*	*Worst Time*
UW Registry **`TakingClass()`** **`PrintSchedule()`** **`MakeClassList()`**				
iterative **`fact()`**				
recursive **`fact()`**				
`PaintFill()`				
linear search				
binary search				