Module -1
Asymptotic Analysis

Tutorial Session

Analyzing Algorithms

Analyze algorithms to gauge:

- Time complexity (running time)
- Space complexity (memory use)

Input size is indicated by a number *n*

- sometimes have multiple inputs, *e.g.* m and n

Running time is a function of *n*

n, n^2, $n \log n$, $18 + 3n(\log n^2) + 5n^3$

RAM Model

RAM (random access machine)

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

Order Notation (Big-O)

Big-O	$T(n) = O(f(n))$ Exist positive constants c, n_0 such that $T(n) \leq cf(n)$ for all $n \geq n_0$	Upper bound
Omega	$T(n) = \Omega(f(n))$ Exist positive constants c, n_0 such that $T(n) \geq cf(n)$ for all $n \geq n_0$	Lower bound
Theta	$T(n) = \theta(f(n))$ $T(n) = O(f(n))$ AND $T(n) = \Omega(f(n))$	Tight bound
little-o	$T(n) = o(f(n))$ $T(n) = O(f(n))$ AND $T(n) != \theta(f(n))$	Strict upper bound

Simplifying with Big-O

By definition, Big-O allows us to:

Eliminate low order terms

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

Eliminate constant coefficients

- $4n \Rightarrow n$
- $0.5\ n \log n \Rightarrow n \log n$
- $\log n^2 = 2 \log n \Rightarrow \log n$
- $\log_3 n = (\log_3 2) \log n \Rightarrow \log n$

But when might constants or low-order terms matter?

Big-O Examples

$n^2 + 100\,n = O(n^2)$

follows from … $(n^2 + 100\,n) \leq 2\,n^2$ for $n \geq 10$

10*10+100*10 <= 2(10*10), n=10

100 < = 100

$n^2 + 100\,n = \Omega(n^2)$

follows from …$(n^2 + 100\,n) \geq 1\,n^2$ for $n \geq 0$

$n^2 + 100\,n = \theta(n^2)$

by definition

$n \log n = O(n^2)$ Upper bound

$n \log n = \theta(n \log n)$ Tight bound

$n \log n = \Omega(n)$ Lower bound

Big-O Usage

Order notation is not symmetric:

- *we can say* $2n^2 + 4n = O(n^2)$
- … *but never* $O(n^2) = 2n^2 + 4n$

Right-hand side is a crudification of the left

Order expressions on left can produce unusual-looking, but *true*, statements:

$O(n^2) = O(n^3)$

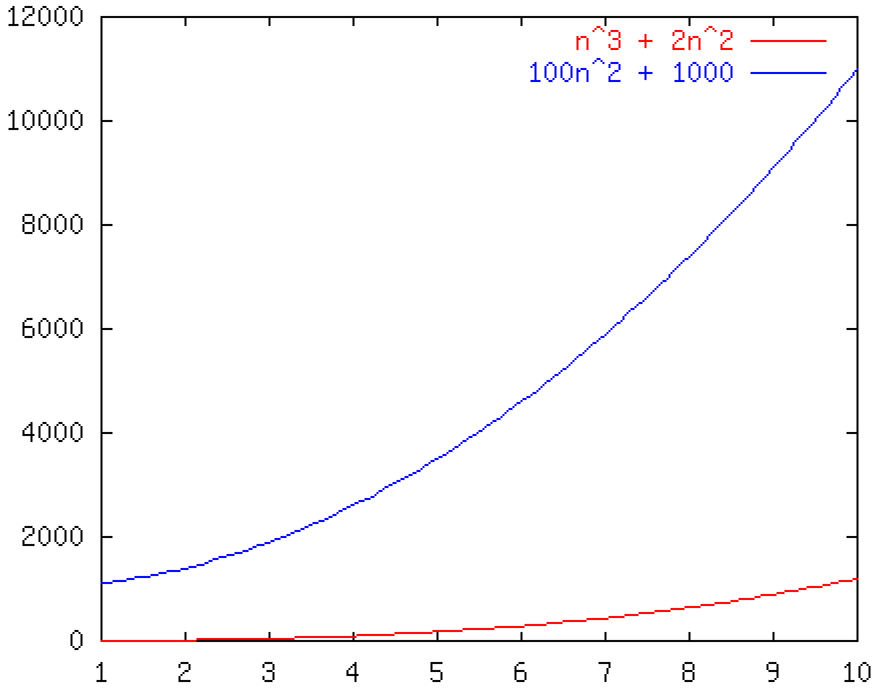
$\Omega(n^3) = \Omega(n^2)$

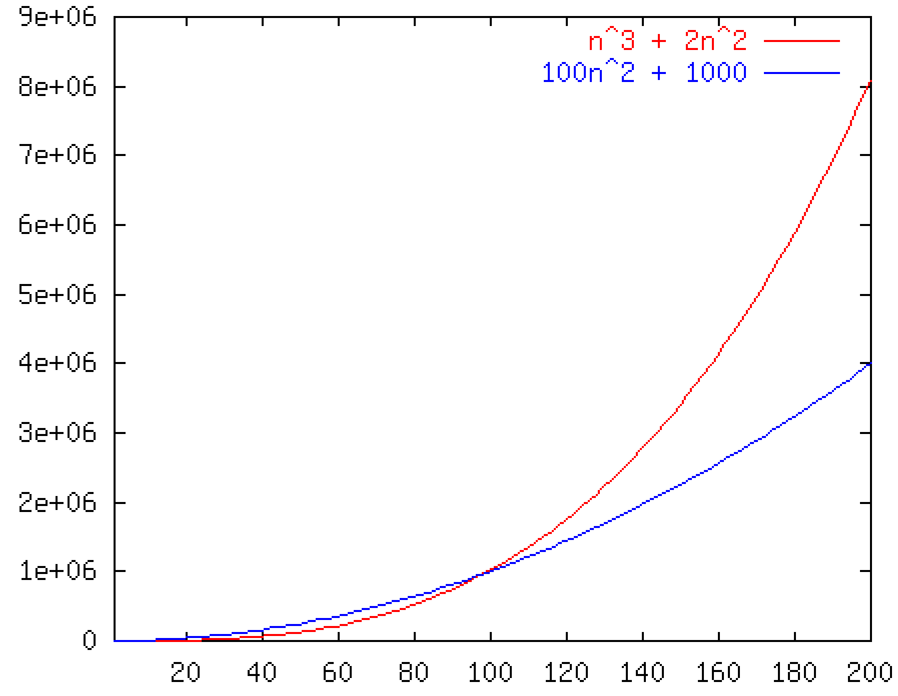
Big-O Comparisons

Function A	vs.	Function #2
$n^3 + 2n^2$		$100n^2 + 1000$
$n^{0.1}$		$\log n$
$n + 100n^{0.1}$		$2n + 10 \log n$
$5n^5$		$n!$
$n^{-15}2^n/100$		$1000n^{15}$
$8^{2\log n}$		$3n^7 + 7n$

Race 1

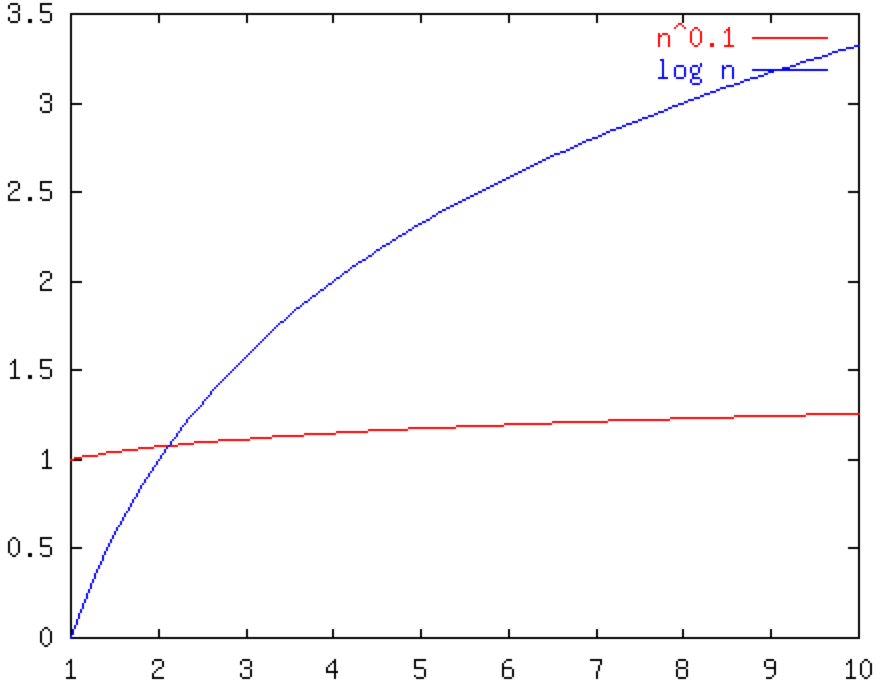
$\mathbf{n^3 + 2n^2}$ vs. $\mathbf{100n^2 + 1000}$

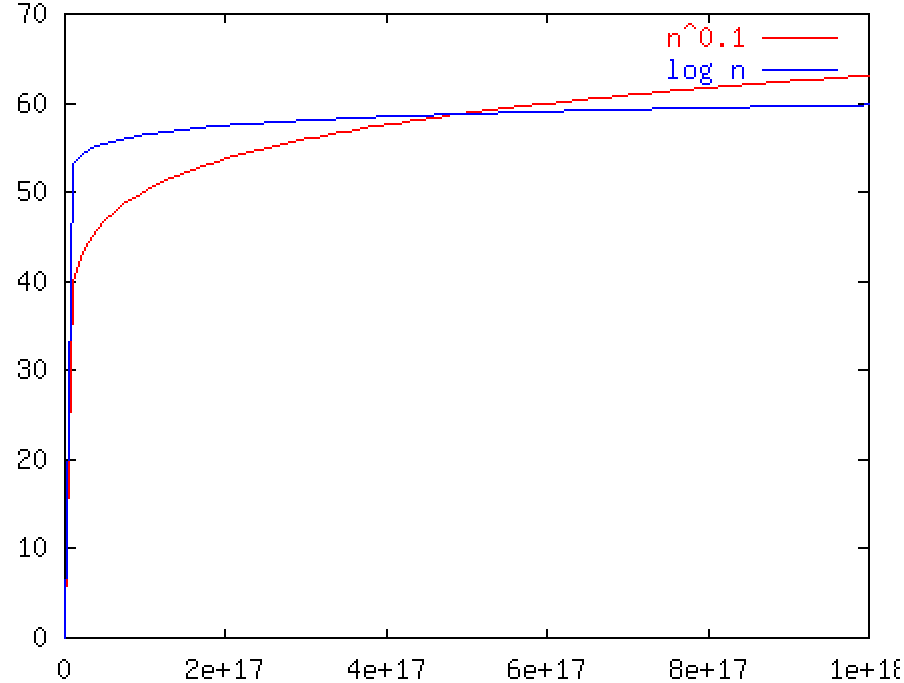




Race 2

$n^{0.1}$ vs. $\log n$

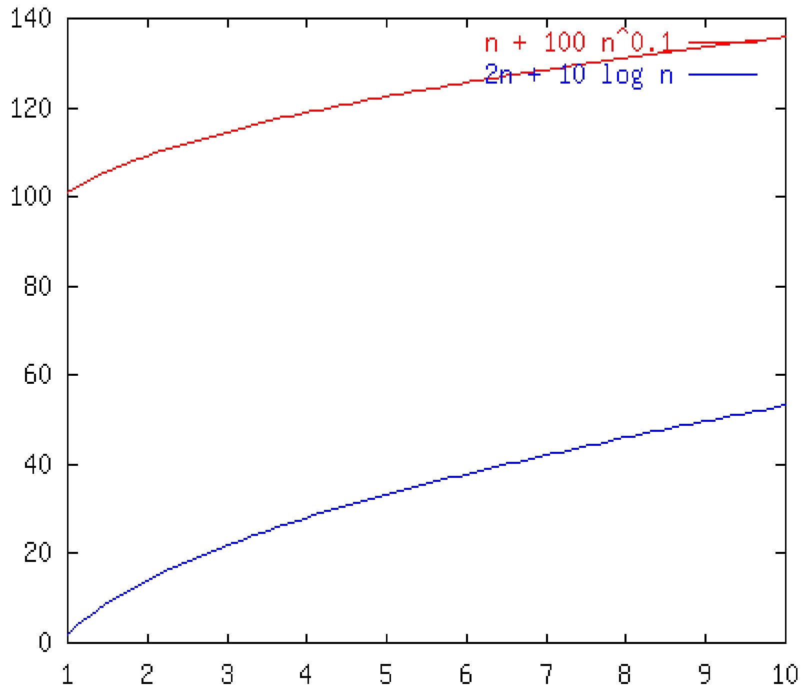


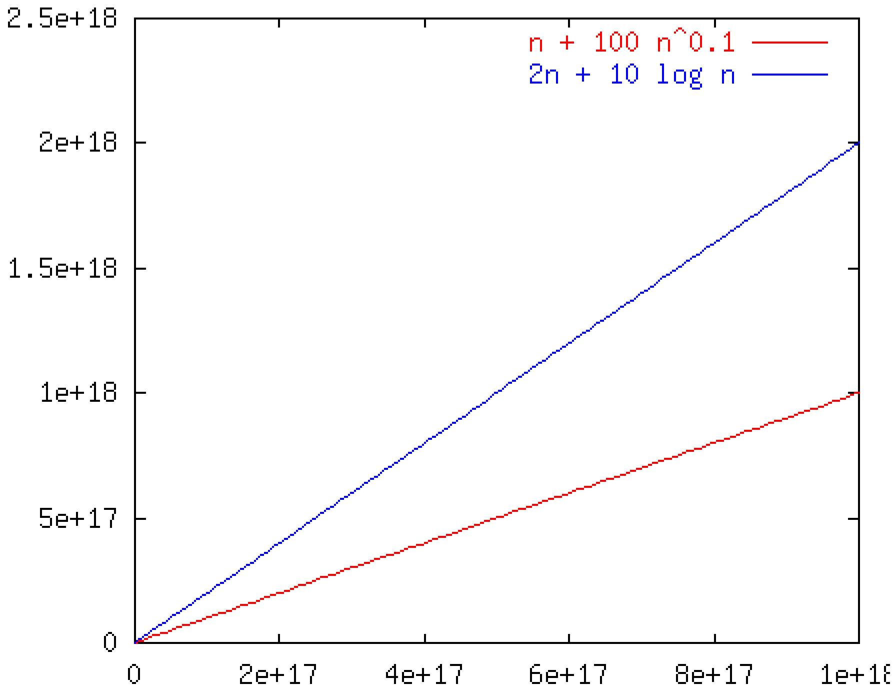


In this one, crossover point is **very late**! So, which algorithm is really better???

Race C

$\mathbf{n + 100n^{0.1}}$ vs. $\mathbf{2n + 10 \log n}$

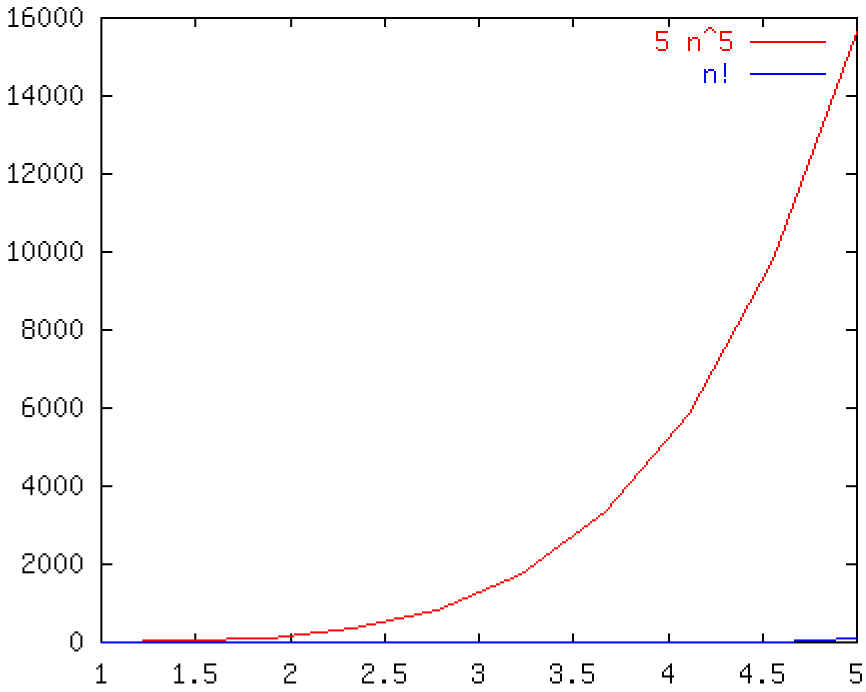


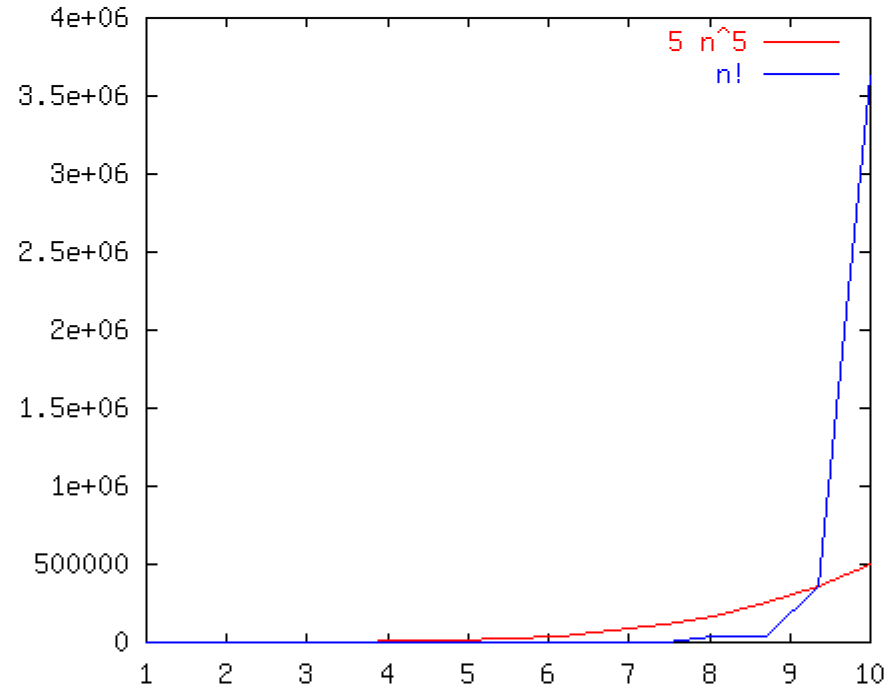


Is the “better” algorithm **asymptotically** better???

Race 4

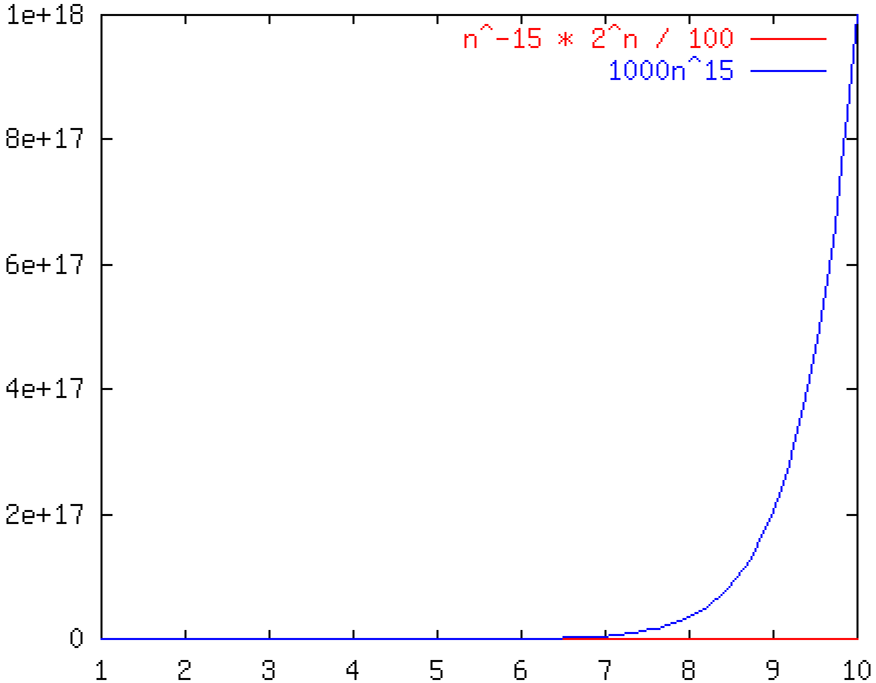
$5n^5$ vs. **$n!$**

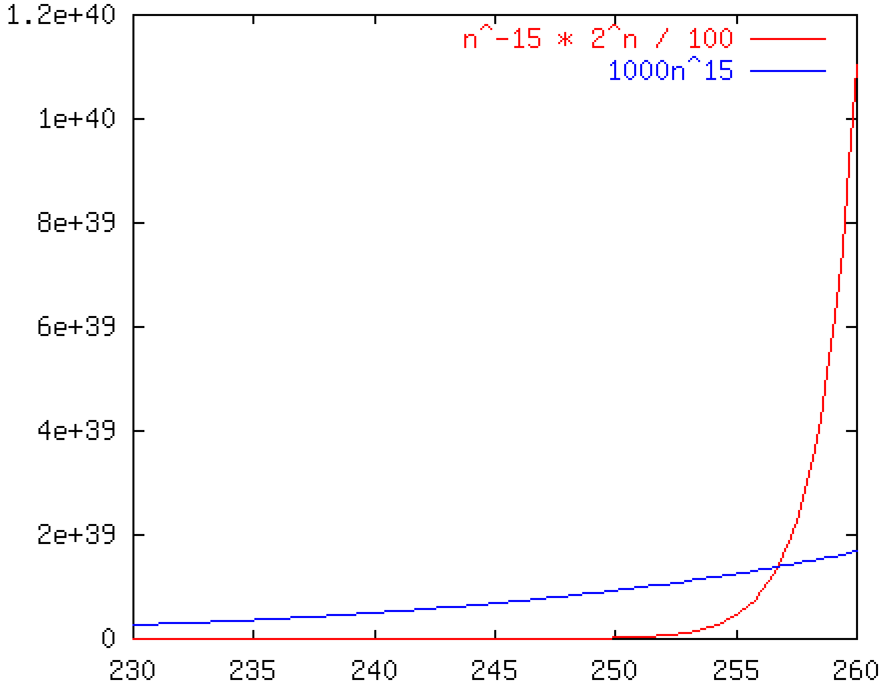




Race 5

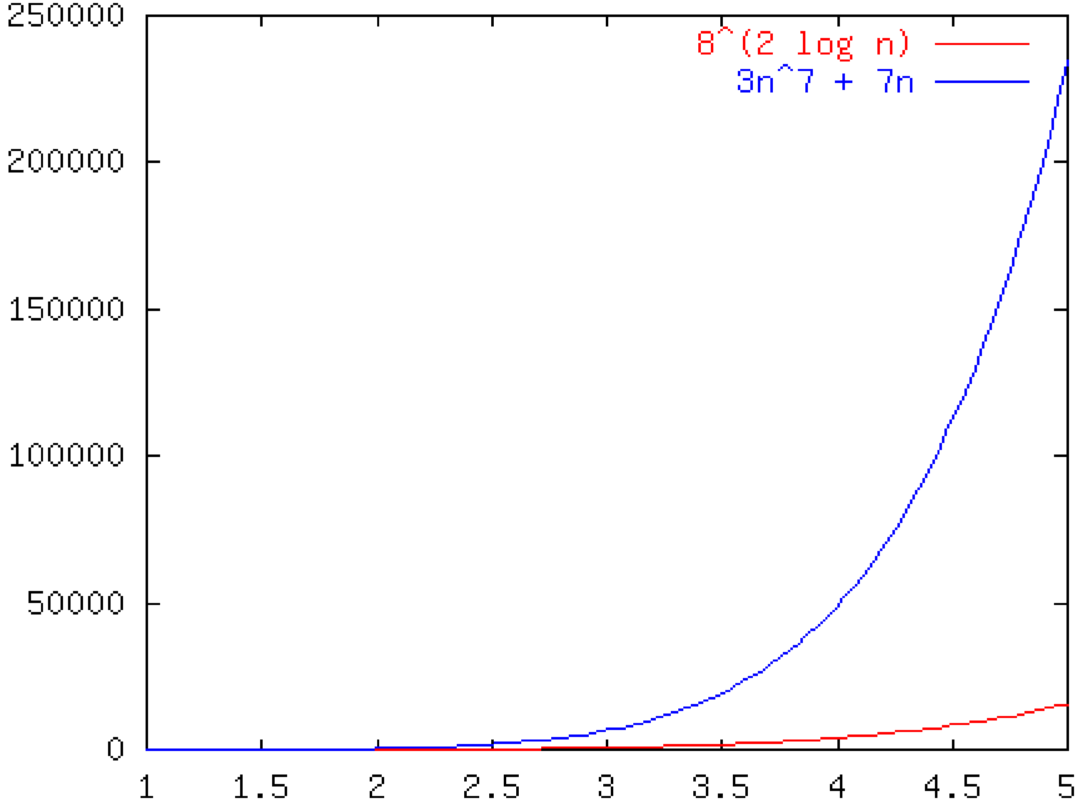
$n^{-15}2^n/100$ vs. $1000n^{15}$





Race VI

$8^{2\log(n)}$ vs. $3n^7 + 7n$



Big-O Winners (i.e. losers)

Function A	vs.	Function #2	Winner
$n^3 + 2n^2$		**$100n^2 + 1000$**	**$O(n^2)$**
$n^{0.1}$		**$\log n$**	**$O(\log n)$**
$n + 100n^{0.1}$		**$2n + 10 \log n$**	**$O(n)$ TIE**
$5n^5$		$n!$	**$O(n^5)$**
$n^{-15}2^n/100$		**$1000n^{15}$**	**$O(n^{15})$**
$8^{2\log n}$		$3n^7 + 7n$	**$O(n^6)$** *why???*

Big-O Common Names

constant:	$O(1)$	
logarithmic:	$O(\log n)$	
linear:	$O(n)$	
log-linear:	$O(n \log n)$	
superlinear:	$O(n^{1+c})$	(c is a constant > 0)
quadratic:	$O(n^2)$	
polynomial:	$O(n^k)$	(k is a constant)
exponential:	$O(c^n)$	(c is a constant > 1)

Kinds of Analysis

Running time may depend on actual input, not just length of input

Distinguish

- Worst case
  - Your worst enemy is choosing input
- Average case
  - Assume probability distribution of inputs
- Amortized
  - Average time over many runs
- Best case (not too useful)

Analyzing Code

C++ operations	constant time
Consecutive stmts	sum of times
Conditionals	larger branch plus test
Loops	sum of iterations
Function calls	cost of function body
Recursive functions	solve recursive equation

Nested Loops

```
for i = 1 to n do
   for j = 1 to n do
      sum = sum + 1
```

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

Dependent Nested Loops

```
for i = 1 to n do
  for j = i to n do
    sum = sum + 1
```

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n}(n+1) - \sum_{i=1}^{n} i =$$

$$n(n+1) - \frac{n(n+1)}{2} = \frac{n(n+1)}{2} \approx n^2$$