Announcements

	MAX	MEAN	MEDIAN	ST DEV
HOMEWORK 1	50	44.2	48	11.5

- Homework 2 due tonight
- Homework 3 out tomorrow
- NO CHEATING

15-150 Fall 2013

Lecture 5

Tuesday, 10 September

Today

- Asymptotic runtime
- Recurrence relations
- Exact and approximate solutions
- Improving efficiency

program $\rightarrow$ recurrence $\rightarrow$ work

asymptotic

- Runtime, for large inputs
- Assume basic ops take *constant time*
- Give big-O classification

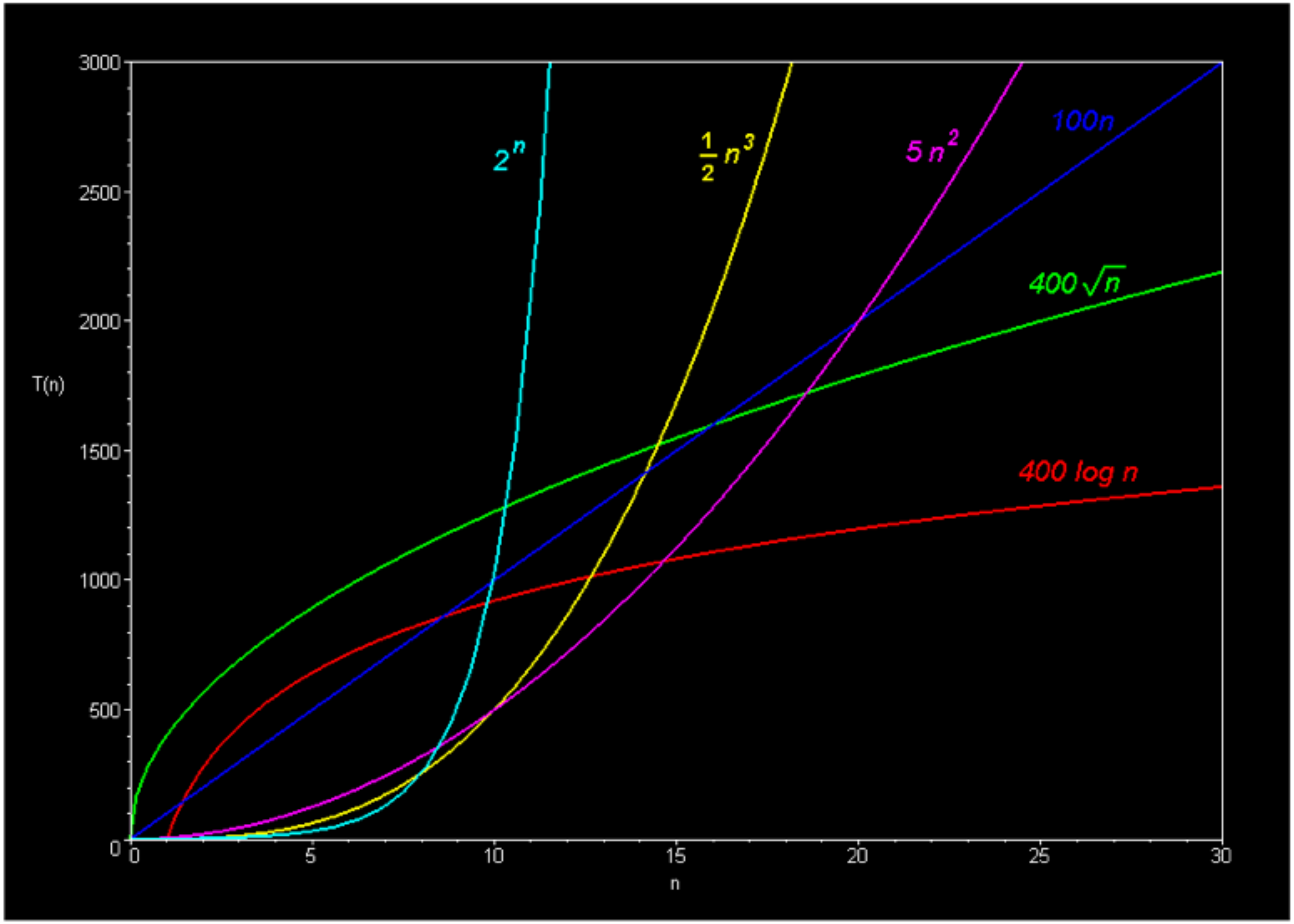
f(n) is O(g(n))

if there are N and c such that

$\forall n \geq N, f(n) \leq c\, g(n)$

The graph below compares the running times of various algorithms.

- Linear -- O(n)
- Quadratic -- O(n^2)
- Cubic -- O(n^3)
- Logarithmic -- O(log n)
- Exponential -- O(2^n)
- Square root -- O(sqrt n)



asymptotic

- Additive constants don't matter

$$n^5+1000000 \quad \text{is } O(n^5)$$

- Multiplicative constants don't matter

$$1000000n^5 \quad \text{is } O(n^5)$$

- Be as accurate as you can

$$O(n^2) \subset O(n^3) \subset O(n^4)$$

- Learn common terminology

logarithmic, linear,
polynomial, exponential

recurrences

- A *recursive function definition* suggests a ***recurrence relation*** for *work*, or *runtime*
  - W(n) = work on inputs of size n
  - Estimates the number of basic operations

base cases

inductive cases

finding solutions

- Try to find a *closed form* solution for W(n) using *induction*
- Find solution to a *simplified* recurrence with the same asymptotic properties
- Appeal to table of standard recurrences

exp

M 4 => if 4=0 then ...
=> 2 * exp (4-1)
=> 2 * M (4-1)
=> 2 * M 3

fun exp (n:int):int =
if n=0 **then** 1 **else** 2 * exp (n-1);

exp 4 =>(1) M 4 =>(4) 2 * (M 3)
=>(4) 2 * (2 * (M 2))
=>(4) 2 * (2 * (2 * (M 1)))
=>(4) 2 * (2 * (2 * (2 * (M 0))))
=>(2) 2 * (2 * (2 * (2 * 1)))
=>(4) 16

where M is (fn n => if n=0 then 1 else 2 * exp(n-1))

exp

- It's not hard to prove that for all $n \geq 0$,

  exp n $=>^{(5n+3)}$ k,
  where k is the numeral for 2^n

But do we need to be so accurate?

And does 5n+3 tell us about actual *runtime* in milliseconds?

No! But it does tell us runtime is ***linear***.

big-O

- It's useful to classify runtimes *asymptotically*
- This abstracts away from additive and multiplicative constants (may be machine-dependent)
- And ignores runtime on small inputs (may be special-cased in the code)

For f, g : int -> int we say that
f is O(g)
if there is a constant c and an integer N
such that for all $n \geq N$,
$|f(n)| \leq c * |g(n)|$.

exp

```
fun exp (n:int):int =
  if n=0 then 1 else 2 * exp (n-1);
```

- Let $W_{exp}(n)$ be the runtime for exp(n)

$$W_{exp}(0) = c_0$$

$$W_{exp}(n) = W_{exp}(n-1) + c_1 \qquad \text{for } n>0$$

for some constants c_0 and c_1

c_0 cost of n=0
c_1 cost of n=0, n-1, mult by 2

solution

- Can prove by induction on n that

$$W_{exp}(n) = c_0 + n\, c_1 \quad \text{for } n \geq 0$$

the work of
exp(n) is *linear in* n

classification

- $W_{exp}(n) = c_0 + n\ c_1$
- $W_{exp}(n)$ is $O(n)$

O-class for $W_{exp}(n)$ independent of c_0, c_1

Let $N=0$, $c = \max(c_0,c_1)+1$

For all $n \geq 0$,

$$W_{exp}(n) \leq c\ n$$

summary

- We've shown that for $n \geq 0$,
  exp n computes the value of 2^n
  in $O(n)$ time
- This fact is *independent* of machine details
  (provided basic operations are constant time)
- Can we do better?

faster exp?

- The definition of exp relies on the fact that

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

- Everybody knows that

$$2^n = (2^{n \text{ div } 2})^2 \qquad \text{if n is even}$$

fastexp

```
fun square(x:int):int = x * x

fun fastexp (n:int):int =
  if n=0 then 1 else
  if n mod 2 = 0 then square(fastexp (n div 2))
                 else  2 * fastexp(n-1)
```

```
fastexp 4 = square(fastexp 2)
          = square(square (fastexp 1))
          = square(square (2 * fastexp 0))
          = square(square (2 * 1))
          = square 4 =16
```

is it faster?

```
fun fastexp (n:int):int =
   if n=0 then 1 else
   if n mod 2 = 0 then square(fastexp (n div 2))
                  else  2 * fastexp(n-1)
```

- Let $W_{fastexp}(n)$ be the runtime for fastexp(n)

$W_{fastexp}(0) = k_0$

$W_{fastexp}(n) = W_{fastexp}(n \text{ div } 2) + k_1$ for $n>0$, even

$W_{fastexp}(n) = W_{fastexp}(n-1) + k_2$ for $n>0$, odd

for some constants k_0, k_1, k_2

is it faster?

```
fun fastexp (n:int):int =
   if n=0 then 1 else
   if n mod 2 = 0 then square(fastexp (n div 2))
                   else  2 * fastexp(n-1)
```

- Let $W_{fastexp}(n)$ be the runtime for fastexp(n)

  $W_{fastexp}(0) = c_0$

  $W_{fastexp}(1) = c_1$

  $W_{fastexp}(n) = W_{fastexp}(n \text{ div } 2) + c_2$ for n>1, even

  $W_{fastexp}(n) = W_{fastexp}(n \text{ div } 2) + c_3$ for n>1, odd

  for some constants c_0, c_1, c_2, c_3

solution?

- Not so obvious how to solve for $W_{fastexp}(n)$
- A closed form would involve c_0, c_1, c_2, c_3
- But we only care about asymptotic behavior
- So we can work with a *simpler* recurrence that has the *same* asymptotic properties

simplification:
choose each constant to be 1

simplification

- Let $T_{fastexp}(n)$ be given by

$$T_{fastexp}(0) = 1$$

$$T_{fastexp}(1) = 1$$

$$T_{fastexp}(n) = T_{fastexp}(n \text{ div } 2) + 1 \quad \text{for } n>1$$

solution

- For n>1, $T_{fastexp}(n)$ *is defined like* log(n)

```
fun log n =
  if n=1 then 0 else log(n div 2) + 1
```

- We know that log(n) = $\log_2(n)$ for all n>0
- Can show that there is a constant c such that

$$T_{fastexp}(n) \leq c \log_2(n)$$

for all large enough n

classification

- $T_{fastexp}(n)$ is $O(\log_2 n)$
- $W_{fastexp}(n)$ depends on c_0, c_1, c_2, c_3
- But we can find constants such that

$$c_{low}\, T_{fastexp}(n) \leq W_{fastexp}(n) \leq c_{high}\, T_{fastexp}(n)$$

and this implies that
$W_{fastexp}(n)$ is also $O(\log_2(n))$

it's faster

- Work of exp(n) is O(n)
- Work of fastexp(n) is O(log n)
- So fastexp is faster than exp, asymptotically

even faster?

- The definition of fastexp relies on

$$2^n = (2^{n \text{ div } 2})^2 \qquad \text{if n is even}$$

$$2^n = 2\,(2^{n-1}) \qquad \text{if n is odd}$$

- A moment's thought tells us that

$$2^n = 2\,(2^{(n \text{ div } 2)})^2 \qquad \text{if n is odd}$$

pow

```
fun pow (n:int):int =
   case n of
       0 => 1
   |  1 => 2
   |  _ => let
                val k = pow(n div 2)
              in
                if n mod 2 = 0 then k*k else 2*k*k
              end
```

work of pow(n)

$W_{pow}(0) = c_0$
$W_{pow}(1) = c_1$
$W_{pow}(n) = c_2 + W_{pow}(n \text{ div } 2)$ for $n>1$

Same recurrence as $W_{fastexp}$

Same asymptotic behavior

pow(n) is O(log n)

comparison

- fastexp(n) and pow(n) have O(log n) work.
- For all n≥0, fastexp(n) = pow(n).
- For all n<0, fastexp(n) fails to terminate,
- For all n<0, pow(n) fails to terminate.
- So fastexp and pow are *extensionally equivalent* and have the same work classification.

badpow

```
fun badpow (n:int):int =
  case n of
     0 => 1
  |  1 => 2
  |  _ => let
            val k2 = badpow(n div 2)*badpow(n div 2)
          in
            if n mod 2 = 0 then k2 else 2*k2
          end
```

work of badpow(n)

$W_{badpow}(0) = c_0$
$W_{badpow}(1) = c_1$
$W_{badpow}(n) = c_2 + 2\,W_{badpow}(n \text{ div } 2)$
for $n>1$

Same asymptotic class as

$T_{badpow}(0) = 1$
$T_{badpow}(1) = 1$
$T_{badpow}(n) = 1 + 2\,T_{badpow}(n \text{ div } 2)$
for $n>1$

examples

$$T_{badpow}(2^0) = 1$$

$$T_{badpow}(2^1) = 1 + 2*T_{badpow}(2^0)$$
$$= 1 + 2*1 = 3$$

$$T_{badpow}(2^2) = 1 + 2*T_{badpow}(2^1)$$
$$= 1 + 2*3 = 7$$

$$T_{badpow}(2^m) = 2^{m+1} - 1$$

analysis

$T_{badpow}(2^m)$ is $O(2^m)$

- $W_{badpow}(2^m)$ is $O(2^m)$
- This implies that $W_{badpow}(n)$ is $O(n)$

$W_{pow}(n)$ is $O(\log n)$
$O(\log n) \subset O(n)$

pow is *asymptotically faster* than badpow

fib

```
fun fib 0 = 1
  |  fib 1 = 1
  |  fib n = fib(n-1) + fib(n-2)
```

$W_{fib}\ 0 = 1$

$W_{fib}\ 1 = 1$

$W_{fib}\ n = 1 + W_{fib}(n-1) + W_{fib}(n-2)$

for $n>1$

analysis

- For all n, $fib(n) \leq W_{fib}(n)$
- $fib(n)$ is $O(\varphi^n)$, where $\varphi \approx 1.618$
- So runtime of fib(n) is *at least* exponential

Surely we can do better!

(to be continued)