# Master Method (3)

## 1 ANNOUNCEMENTS

Homework 1

- due Tues. Sep 16th.
- follow the homework link on the schedule page

## 2 A NORMAL FORM

### 2.1 Putting Bounds in Simplest Form

This is not standard, but I think you might find it useful.
Many of the most important functions are asymptotically equivalent to a function that can be specified in the following "normal" form:

- $(n)^{a_0}(\log n)^{a_1}(\log\log n)^{a_2}\ldots(\log^{(k)}(n))^{a_k}$

where $a_i \geq 0$ for all $0 \leq i < k$, and $a_k > 0$.
Any function that combines addition, multiplication, logarithm, maximization, or minimization of this type of function can be reduced to this form. Ought to be able to handle division also.
Scalar Multiplication Rule: As an easy starting point, any positive leading multiplicative constants can be dropped: $6n\log n \to n\log n$.

### 2.2 Addition Rule

Here's how to add two functions in normal form:

- $(n)^{a_0}(\log n)^{a_1}(\log\log n)^{a_2}\ldots(\log^{(k)}(n))^{a_k}$
- $+(n)^{b_0}(\log n)^{b_1}(\log\log n)^{b_2}\ldots(\log^{(k)}(n))^{b_k}$
- $=(n)^{c_0}(\log n)^{c_1}(\log\log n)^{c_2}\ldots(\log^{(k)}(n))^{c_k}$

where $c_i = a_i$ unless $b_i > a_i$ and $a_j = b_j$ for all $0 \leq j < i$ (in which case $c_i = b_i$). Also, technically, we might need to pad out either $a$ or $b$ so they are the same length and then drop the trailing zeros from $c$.
Example:

- $n \log n + \sqrt{n}(\log \log n)^2$
- $a_0 = 1$, $a_1 = 1$, $a_2 = 0$, $b_0 = 1/2$, $b_1 = 0$, $b_2 = 2$.
- Therefore, $c_0 = 1$, $c_1 = 1$, $c_2 = 0$.
- So, $n \log n + \sqrt{n}(\log \log n)^2 = \Theta(n \log n)$.

Idea: Whichever function has the largest power of $n$ wins, with ties broken by the largest power of $\log n$, then $\log \log n$, and so on.
Note: Maximization is the same.

## 2.3 Multiplication Rule

Here's how to multiply two functions in normal form:

- $(n)^{a_0}(\log n)^{a_1}(\log \log n)^{a_2} \ldots (\log^{(k)}(n))^{a_k}$
- $\times (n)^{b_0}(\log n)^{b_1}(\log \log n)^{b_2} \ldots (\log^{(k)}(n))^{b_k}$
- $= (n)^{c_0}(\log n)^{c_1}(\log \log n)^{c_2} \ldots (\log^{(k)}(n))^{c_k}$

where $c_i = a_i + b_i$.
Example:

- $n \log n \times \sqrt{n}(\log \log)^2 n$
- $a_0 = 1$, $a_1 = 1$, $a_2 = 0$, $b_0 = 1/2$, $b_1 = 0$, $b_2 = 2$.
- Therefore, $c_0 = 3/2$, $c_1 = 1$, $c_2 = 2$.
- So, $n \log n \times \sqrt{n}(\log \log)^2 n = \Theta(n^{3/2} \log n (\log \log n)^2)$.

Idea: Add the corresponding exponents.

## 2.4 More Rules

It is also possible to derive similar rules for minimum of two functions in normal form and the logarithm of a single function in normal form. Please do so for homework.

## 2.5 Summary

We talked about how to classify running-time functions by their asymptotic growth. We also discussed how to express a function in its simplest form.
These are tools that are invaluable for analyzing all sorts of algorithms.
Today we'll connect asymptotic growth to another important tool: recurrence relations. First, some more context.

# 3 SENSE OF SCALE

## 3.1 Preview

Here are examples of some of the functions we care about:

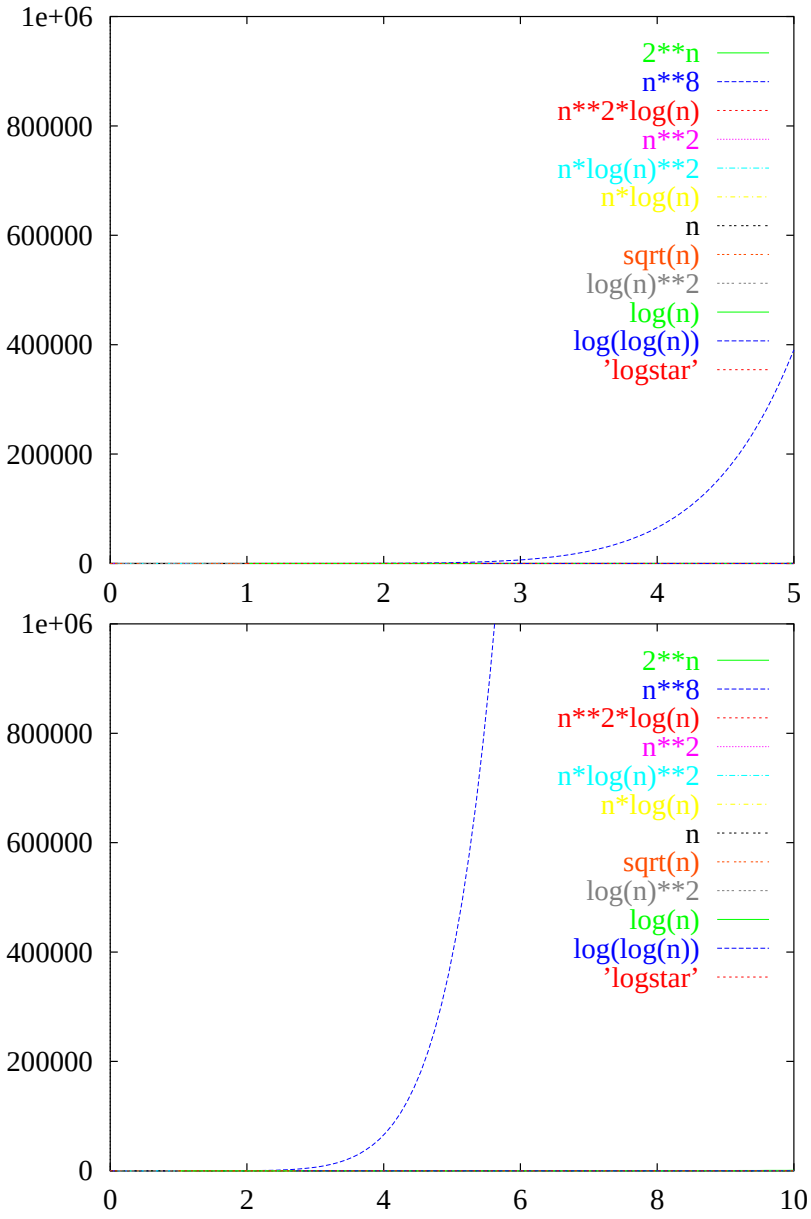
| | |
|---|---|
| $2^n$ | $n$ |
| $n^8$ | $\sqrt{n}$ |
| $n^2 \log(n)$ | $\log^2(n)$ |
| $n^2$ | $\log(n)$ |
| $n \log^2(n)$ | $\log(\log(n))$ |
| $n \log n$ | $\log^*(n)$ |

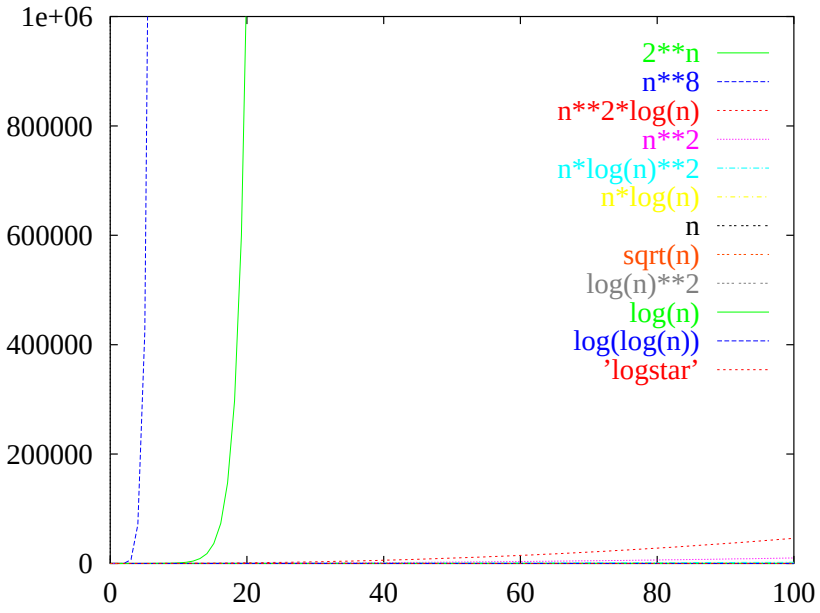
## 3.2 How They Scale

If an algorithm has running time $g(n)$, what size problems can we solve if we can wait a million units?
We'll plot all the $g(n)$s for various $n$ on a scale of one to a million.
(Inspired by the short film "Powers of Ten," by Charles and Ray Eames.)

## 3.3 Heavy Hitters

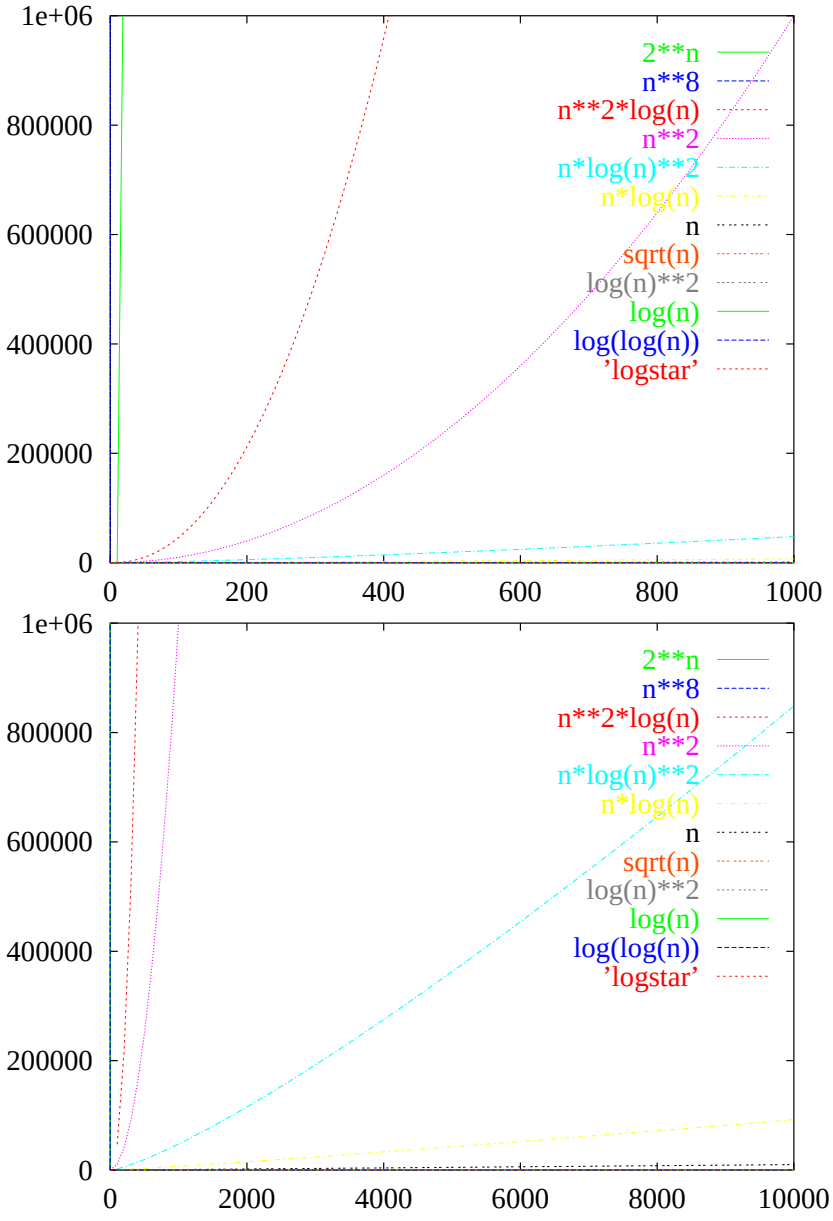
First, we see the very fast growing functions exploding away, high degree polynomials, exponentials. Note that these two curves cross somewhere between 43 and 44 (at a height of about $10^{13}$).


1e+06
800000
600000
400000
200000
0
0
1
2
3
4
5
2**n
n**8
n**2*log(n)
n**2
n*log(n)**2
n*log(n)
n
sqrt(n)
log(n)**2
log(n)
log(log(n))
'logstar'
1e+06
800000
600000
400000
200000
0
0
2
4
6
8
10
2**n
n**8
n**2*log(n)
n**2
n*log(n)**2
n*log(n)
n
sqrt(n)
log(n)**2
log(n)
log(log(n))
'logstar'


1e+06
800000
600000
400000
200000
0
0
20
40
60
80
100
2**n
n**8
n**2*log(n)
n**2
n*log(n)**2
n*log(n)
n
sqrt(n)
log(n)**2
log(n)
log(log(n))
'logstar'

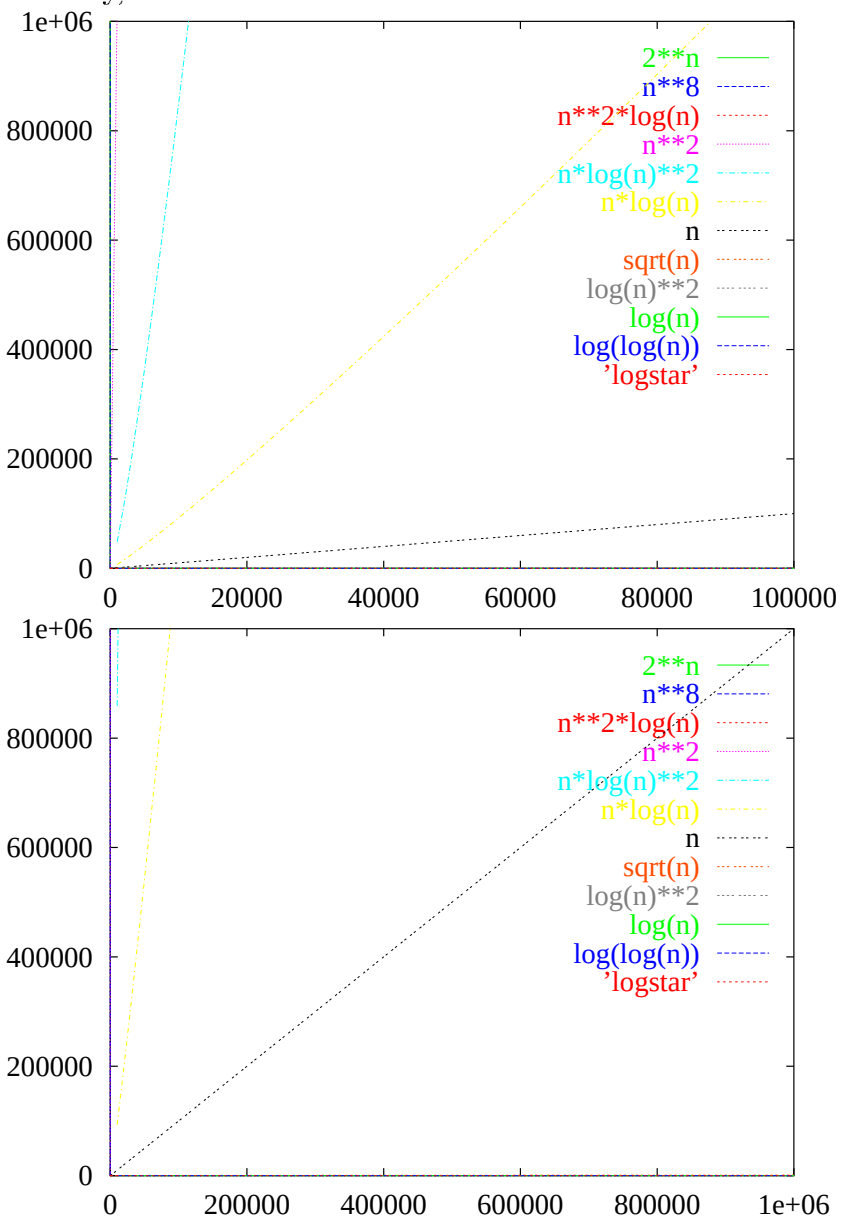
## 3.4 Superlinear

Next, we lose the functions that functions that grow faster than linear. Note that $n \log n$ hangs in their quite a while.



## 3.5 Linear

Eventually, even linear starts to have trouble.

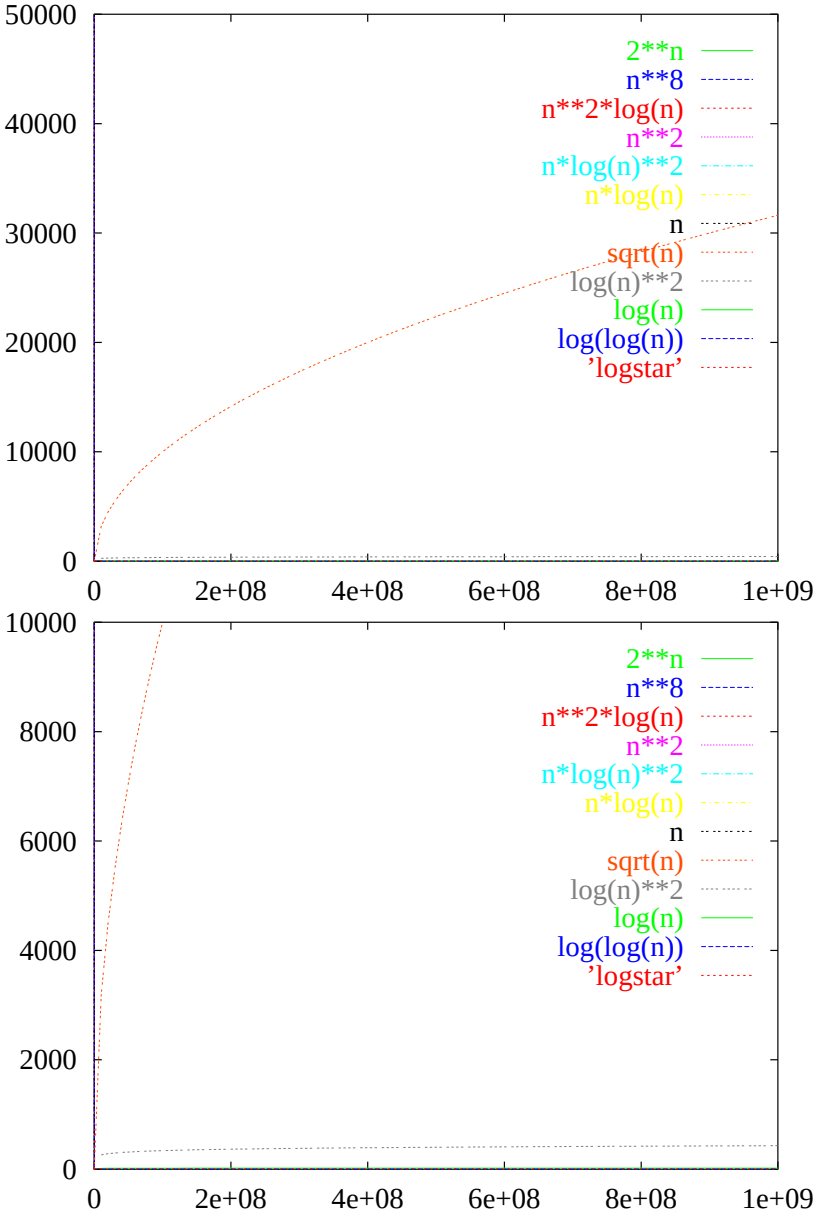


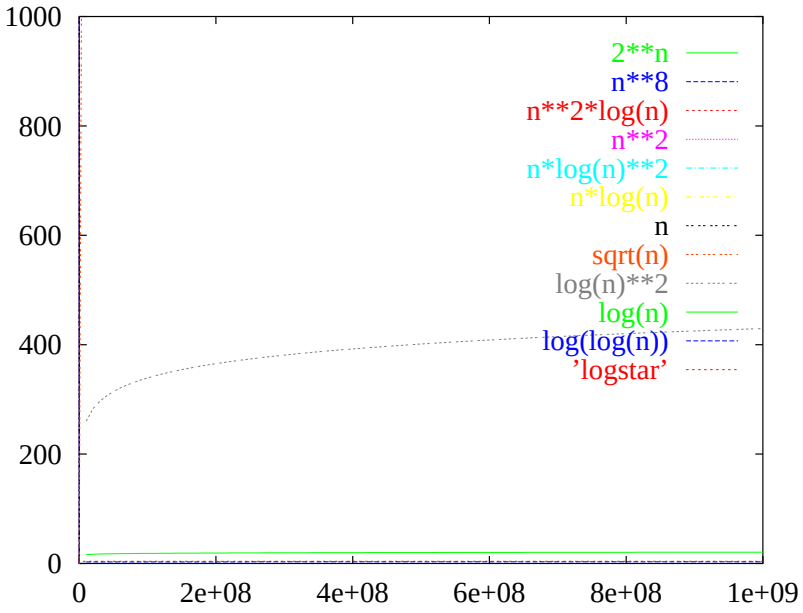
## 3.6 Zooming In

Now that the "slow" algorithms are out of the way, we zoom in on the $y$ axis so we can see the distinctions more clearly.
That is, we will look at problem sizes of a billion, but tighten our notion of how long we are willing to wait for an answer.

## 3.7 Superlog

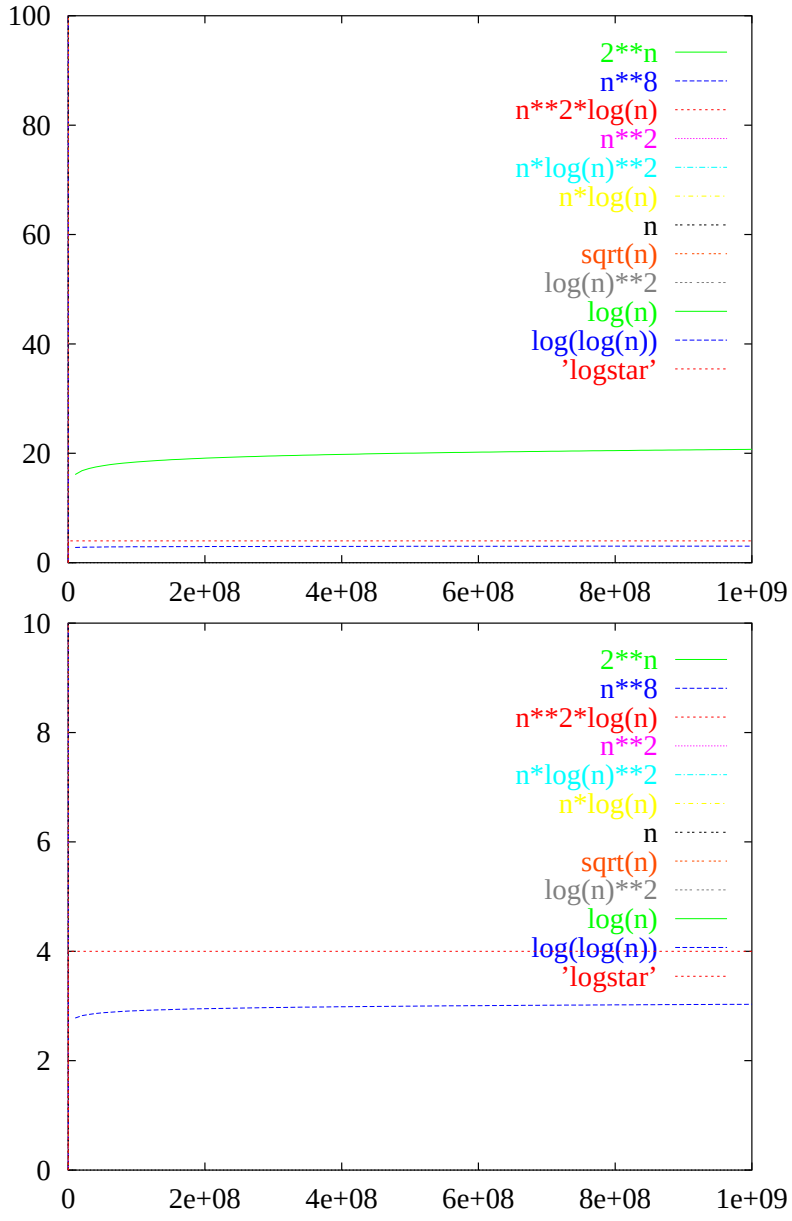
Square root and log squared are the next to go.




1000
800
600
400
200
0
0
2e+08
4e+08
6e+08
8e+08
1e+09
2**n
n**8
n**2*log(n)
n**2
n*log(n)**2
n*log(n)
n
sqrt(n)
log(n)**2
log(n)
log(log(n))
'logstar'

## 3.8 Log and Below

And finally, log, log log, and log star. Note that these two curves cross somewhere past $n = 10^{24}$ and before $10^{64}$ at a height of about 5.



# 4 SOLVING RECURRENCES

## 4.1 The Substitution Method

Idea: Make an intelligent guess and prove by induction.
Example:

- Solve: $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- Guess: $T(n) \leq cn \log n$.
- Prove:
  - Base case: Assume constant size inputs take constant time.
  - Assume: $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$
  - Then:

$$\begin{aligned} T(n) &\leq 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) + n \\ &\leq 2c(n/2)\log(n/2) + n \\ &= cn\log(n/2) + n \\ &= cn\log(n) - cn + n \\ &= cn\log(n) - n(c-1) \\ &\leq cn\log(n) \end{aligned}$$

  - So we see we need $c \geq 1$.

## 4.2 Making Good Guesses

Start with the answer to something similar you've seen before.
Use common sense (?) to figure out which aspects of a formula aren't relevant.
For example, in $T(n) = 2T(n/2 + 17) + n$, the 17 is probably not that important since $n/2 + 17$ is a lot like $n/2$ for large $n$.
The book lists a number of subtleties that crop up and heuristics to help get you out of trouble.

## 4.3 The Iteration Method

The subtitution method is great once you've gotten enough experience to make a good guess at the answer.
The iteration method is a nice way to gain that experience.

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= 3T(n/4) + n \\
&= 3(3T(n/16) + n/4) + n = 9T(n/16) + 7n/4 \\
&= 9(3T(n/64) + n/16) + 7n/4 = 27T(n/64) + 37n/16 \\
&= 27(3T(n/256) + n/64) + 37n/16 = 81T(n/256) + 175n/64 \\
&= 3^4 T(n/(4^4)) + n\sum_{i=0}^{3}(3/4)^i \\
&= \ldots \\
&= 3^k T(n/(4^k)) + n\sum_{i=0}^{k-1}(3/4)^i \\
&= 3^k T(n/(4^k)) + 4n(1-(3/4)^k)
\end{aligned}
$$

## 4.4 Terminating the Iteration

$T(n) = 3^k T(n/(4^k)) + 4n(1-(3/4)^k)$.
For what $k$ does $n/(4^k) = 1$?

$$
\begin{aligned}
n/(4^k) &= 1 \\
n &= (4^k) \\
\log(n) &= k\log(4) \\
k &= \log(n)/2
\end{aligned}
$$

Substituting into the formula above...

- $3^k = 3^{\log(n)/2} = n^{\log(3)/2}$
- $(3/4)^k = (3/4)^{\log(n)/2} = n^{\log(3)/2-1}$
- So,

$$
\begin{aligned}
T(n) &= n^{\log(3)/2}T(1) + 4n(1-n^{\log(3)/2-1}) \\
&= n^{\log(3)/2} + 4n - 4n^{\log(3)/2} \\
&= 4n - 3n^{\log(3)/2} \\
&= O(n)
\end{aligned}
$$

## 4.5 The Table Method

The iteration method is pretty algebra intensive.

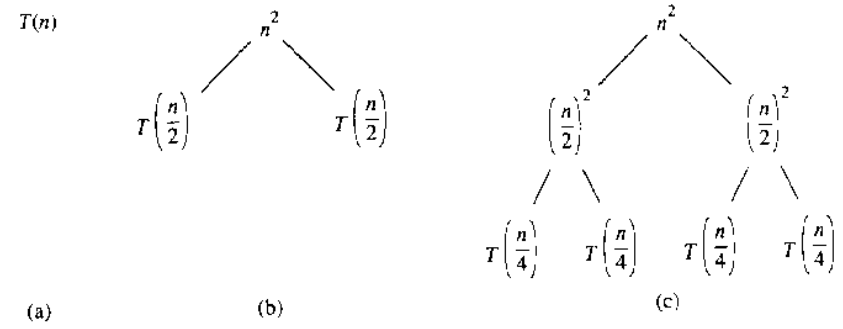
$$\begin{aligned} T(0) &= 3 \\ T(1) &= 3\sqrt{2} \\ T(n) &= 2T(n-2) \end{aligned}$$

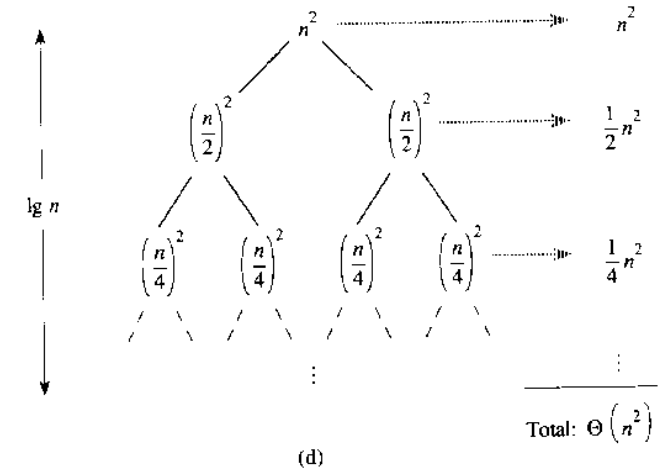
| $n$ | $T(n)$ | |
|---|---|---|
| 0 | 3 | |
| 1 | $3\sqrt{2}$ | |
| 2 | 6 | $3(2)$ |
| 3 | $6\sqrt{2}$ | |
| 4 | 12 | $3(2^2)$ |
| 5 | $12\sqrt{2}$ | |
| 6 | 24 | $3(2^3)$ |
| 7 | $24\sqrt{2}$ | |

If n is even, $T(n) = 3(2^{n/2})$.
Now check $n$ odd... hey, it works!

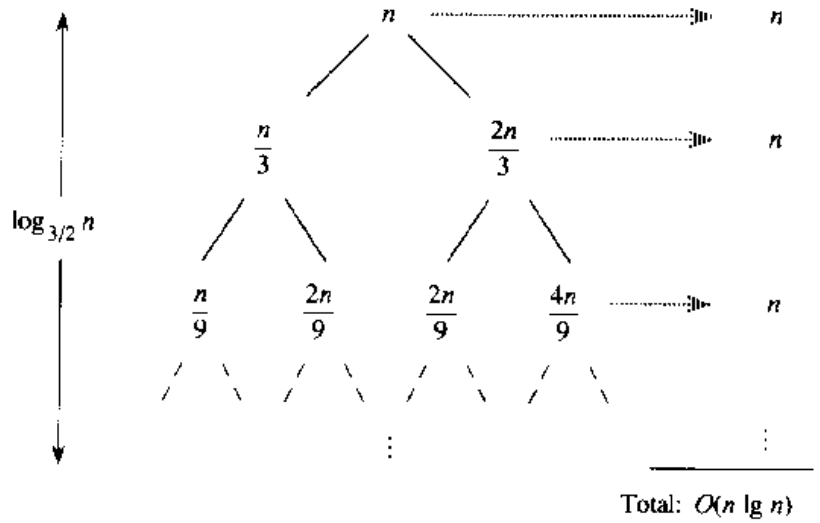
## 4.6 Recursion Tree

Starting to get closer to an "automated" approach.
Take $T(n) = 2T(n/2) + n^2$.
Imagine recursion unfolding as a tree.




A harder version: $T(n) = T(n/3) + T(2n/3) + n$.




## 4.7 Summary

There are a number of ways of attacking recurrences. It takes a certain amount of experience to use them to solve hard recurrences.

We'll look at a very nice tool for solving a wide variety of recurrences, but first, a catalog of some important ones.

# 5 CATALOG OF RECURRENCES

## 5.1 Introduction

Let's look at these functions (and some others), as they appear when expressed as recurrence relations.
Try to picture the recurrence trees for these!

## 5.2 Linear

- function: $\Theta(n)$.
- recurrence: $T(1) = 1$, $T(n) = 1 + T(n-1)$
- context: Singly-nested loops, visit everything once.
- examples: Naive multiplication, depth-first search.
- variations: $T(n) = 1+2T(n/2)$, $T(n) = n+T(n/2)$, $T(n) = T(n/5)+T(7n/10+6)+n$.

## 5.3 Log

- function: $\Theta(\log n)$.
- recurrence: $T(1) = 1$, $T(n) = 1 + T(n/2)$
- context: Recurse on half of input and throw half away.
- examples: Euclid, Repeated Squaring, search in balanced trees.
- variations: $T(n) = 1 + T(99n/100)$, $T(n) = T(2n/3) + n/3$.

## 5.4 Quadratic

- function: $\Theta(n^2)$.
- recurrence: $T(1) = 1$, $T(n) = n + T(n-1)$
- context: Nested loops.
- examples: Matrix multiplication, sorting.
- variations: $T(n) = 1+T(99n/100)+T(n/100)$, other polynomials: $T(n) = n^2+T(n-1)$.

## 5.5 Exponential

- function: $\Theta(2^n)$.
- recurrence: $T(1) = 1$, $T(n) = 2T(n-1)$
- context: Solve both of two slightly smaller versions of problem.
- examples: Number of nodes in a binary tree.
- variations: $T(n) = 1 + 2T(n-1)$, other exponentials: $T(n) = kT(n-1)$.

## 5.6 N Log N

- function: $\Theta(n \log n)$.
- recurrence: $T(1) = 1$, $T(n) = n + 2T(n/2)$
- context: Scan, divide, recurse on both halves.
- examples: Sorting, Fast Fourier Transform.
- variations: $T(n) = \log n + T(n-1)$.

# 6 MASTER METHOD

## 6.1 Basic Idea

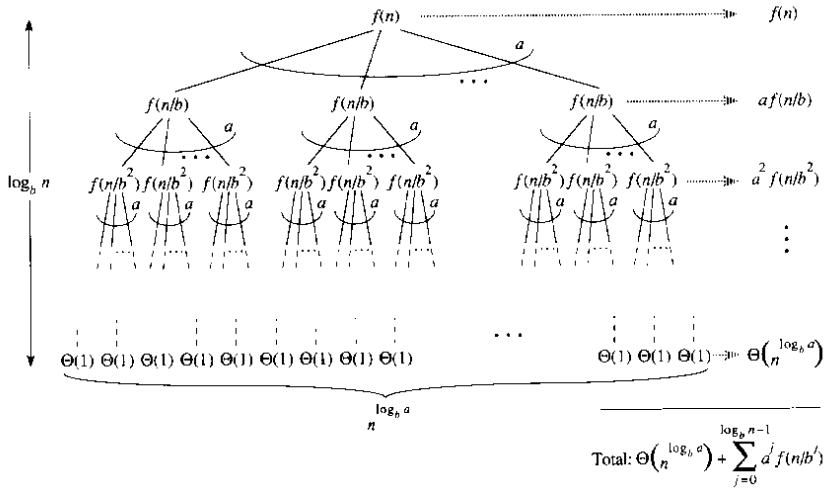
Handles recurrences of the form: $T(n) = aT(n/b) + f(n)$. where $a \geq 1$, $b > 1$, and $f(n)$ is some function.
We gloss over some technicalities:

- $T(n)$ is usually only defined for integer $n$, so we really need $T(\lceil n/b \rceil)$.
- boundary conditions like $T(1) = 0$.

This is ok because these details get swallowed up by the big O or big Theta that we are eventually going to wrap around things anyway.

## 6.2 The Master Tree

$a$ is branching factor, $b$ determines how deep the tree goes, $f(n)$ determines the weight of each level.




## 6.3 The Theorem

We are given the recurrence $T(n) = aT(n/b) + f(n)$.

1. If $f(n) = O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b(a)})$.
2. If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \log n)$.
3. If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.
4. If $f(n) = \Theta(n^{\log_b(a)} \log^k(n))$ for $k \geq 0$, then $T(n) = \Theta(n^{\log_b(a)} \log^{k+1} n)$.

## 6.4 Justification

The basic justification is very simple.
$n^{\log_b(a)}$ is the number of leaves on the tree.

- If $f(n)$ polynomially less than $n^{\log_b(a)}$, then the number of leaves dominates in the recurrence: $\Theta(n^{\log_b(a)})$ (Rule 1).
- If $f(n)$ polynomially more than than $n^{\log_b(a)}$, then the first appearance of $f(n)$ in the tree dominates in the recurrence: $\Theta(f(n))$ (Rule 3). (There is a "regularity" condition on $f(n)$ for this to be true that holds for nearly all functions that we'll see, so you can ignore it).
- If $f(n)$ and $n^{\log_b(a)}$ balance, then $f(n)$ makes a contribution at every level: $\Theta(f(n) \log n)$ (Rule 2).

## 6.5 The Gaps

Note that there is a polynomial gap on either side. It's not enough for $f(n)$ to be bigger (or smaller) than $n^{\log_b(a)}$, it has to be polynomially so (i.e., the $\epsilon > 0$ matters).
In one case, we can partially fill this gap. That is, if $f(n)$ is just a log factor bigger than $n^{\log_b(a)}$, we still get a contribution from each level: $T(n) = \Theta(n^{\log_b(a)} \log^{k+1} n)$.

## 6.6 Examples

$T(n) = 9T(n/3) + n$

- $a = 9$, $b = 3$, $f(n) = n$, $n^{\log_b(a)} = n^2$. $n^2$ is polynomially bigger.
- So, $T(n) = \Theta(n^2)$.

$T(n) = 3T(n/7) + n^{\frac{3}{4}}$

- $a = 3$, $b = 7$, $f(n) = n^{\frac{3}{4}} = n^{0.75}$, $n^{\log_b(a)} \approx n^{0.56}$. $n^{0.75}$ is polynomially bigger.
- So, $T(n) = \Theta(n^{\frac{3}{4}})$.

Note: As long as $f(n) = \Theta(n^k)$, for $k \geq 0$, Master Theorem definitely applies!
$T(n) = T(4n/5) + \log^2 n$

- $a = 1$, $b = 5/4$, $f(n) = \log^2 n$, $n^{\log_b(a)} = 1$. $f(n)$ is not polynomially bigger. But Rule 4 applies (with $k = 2$).
- So, $T(n) = \Theta(\log^3(n))$.

$T(n) = 2T(n/4) + \sqrt{n} \log\log(n)$

- $a = 2$, $b = 4$, $f(n) = \sqrt{n} \log\log n$, $n^{\log_b(a)} = n^{1/2}$, so $f(n)$ is not polynomially larger than $n^{\log_b(a)}$.
- So, Master Theorem does not apply.