Announcements

- HW due Wed. at start of class

- Next HW out Wed, due the following Friday
For any induction proof:

4 pieces:
- know what you are inducting on
- base case
- Inductive Hypothesis
- Inductive Step
**Ex: The Gossip Problem**

- There are $n$ people, and each knows a unique secret.
- Every time 2 people call each other, they tell all of the secrets that they know to each other.

How many phone calls are necessary before everyone knows all the secrets?
Thm: If $n \geq 4$, then $2n-4$ calls are enough.

Proof: Induction on # of people

Base case: 4 people

Goal: $2 \cdot 4 - 4 = 4$ calls

$X\xrightarrow{1,33} \xrightarrow{1,33} \xrightarrow{1 + 3 \text{ talk}} \xrightarrow{2 \cdot 4 \text{ talk}} \xrightarrow{1 + 2 \text{ talk}} \xrightarrow{3 \cdot 4 \text{ talk}}$
III: For k<n people, 2k-4 calls suffice.
IS: n people n calls take person n away
n-1

2(n-1)-4 = 2n-6
n calls 1
Now, recursion!

Induction starts at bottom & builds up.
Recursion is the natural dual idea:
- Start with *n* things
- Reduce to smaller subproblem(s)
- Eventually stop at some small base case
Solving recurrences

\[ H(n) = 2H(n-1) + 1 \]
\[ M(n) = 2M\left(\frac{n}{2}\right) + n \]
\[ T(n) = T\left(\frac{3n}{4}\right) + n \]

How to solve?
- Unrolling
- Master theorem: \( S(n) = aS\left(\frac{n}{b}\right) + f(n) \)
- Guess & check
- Characteristic equation method
  (Annihilator method)
Recursion

Based on the idea of reduction: reducing $X$ to $Y$ means giving an algorithm to solve $X$ which may use $Y$ as a subroutine.

Ex: (from last class)

the Congress partitioning by used priority queue
Recursion (cont)

Reduce a problem to a simpler instance of the same problem.

Necessary pieces (like induction):
- base case
- recursive call (to smaller input)
- (some extra work)
Towers of Hanoi

Rules:
- no
- move 1 at a time
How?
Think recursively!
Base case?
The trick: think recursively!

Most people try to unroll the recurrence - I think it's not necessary!
Think of this like a procedure:

```
HANOI(n, src, dst, tmp):
if n > 0
    HANOI(n - 1, src, tmp, dst)
    move disk n from src to dst
    HANOI(n - 1, tmp, dst, src)
```

1 3 2
Proof of correctness

Induction on # of disks, $n$:

Base case $n = 1$, recursive calls do nothing + 1 disk moves correctly

IH: For $k < n$ disks, alg. is correct.

IP: $n$ disks.

by IH, $n-1$ top move to temp

by IH, one goes to dest.

by IH, $n-1$ top move correctly to dest.

(don't violate rules in top level)
Run time: Let \( H(k) \) = time to solve towers of Hanoi with \( k \) pancakes

\[
\begin{align*}
H(0) &= 0 \\
H(1) &= 1 \\
H(2) &= 3 \\
H(n) &= H(n-1) + 1 + H(n-1) \\
H(n) &= 2H(n-1) + 1
\end{align*}
\]

\[ a_n = 2a_{n-1} + 1 \]
\[ x - 2 = 0 \]
\[ x = 2 \]

\[ \rho(n) = 1 \quad \text{degree} = 0 \]

\[ H(n) = c_1 \cdot 2^n + c_2 \]
\[ = 2^n - 1 \quad (?) \]
Another (old) example: Merge Sort

According to Knuth, suggested by von Neumann around 1945.

Idea:
1. Subdivide array into 2 parts.
2. Recursively sort the 2 parts.
3. Merge them back together.

<table>
<thead>
<tr>
<th>Input:</th>
<th>S O R T I N G E X A M P L E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide:</td>
<td>S O R T I N G E X A M P L E</td>
</tr>
<tr>
<td>Recurse:</td>
<td>I N O S R T</td>
</tr>
<tr>
<td>Merge:</td>
<td>A E G I L M N O P S R T X</td>
</tr>
</tbody>
</table>
Keys: If thinking recursively, only step 3 is non-trivial!

```
MERGE-SORT(A[1..n]):
  if (n > 1)
    m ← ⌊n/2⌋
    MERGE-SORT(A[1..m])
    MERGE-SORT(A[m+1..n])
    MERGE(A[1..n], m)
```

(Again, avoid unrolling.)

What’s my base case here?

Size 1 (or less)
How to merge?

Input: SORTINGEXAMPLE
Divide: SORTINGEXAMPLE
Recurse: INOSRTAEGLMPX
Merge: AEGLIMNOPSTRX
Write a subroutine:

```
MERGE(A[1..n], m):
    i ← 1; j ← m + 1
    for k ← 1 to n
        if j > n
            B[k] ← A[i]; i ← i + 1
        else if i > m
            B[k] ← A[j]; j ← j + 1
        else if A[i] < A[j]
            B[k] ← A[i]; i ← i + 1
        else
            B[k] ← A[j]; j ← j + 1
    for k ← 1 to n
        A[k] ← B[k]
```
Proof of correctness: Actually, 2 of them.

Lemma: MERGE results in sorted order.

Runtime: