Exercise

• Form groups of 5 people (split rows in half)
• Person sitting in middle is note-taker
• Share the lists of steps for analyzing a recursive procedure. Come up with a revised list combining best ideas. (5 minutes)
• Note-taker: copy list on a transparency.
• Then: use your method to analyze the following procedure. (10 minutes)
• Note-taker: copy solution on a transparency

How I Analyze a Recursive Program

1. Write recursive equation, using constants a, b, etc.
2. Expand the equation repeatedly, until I can see the pattern
3. Write the equation that captures the pattern – make an inductive leap! – in terms of a new variable k
4. Select a particular value for the variable k in terms of n – pick a value that will make the recursive function a constant
5. Simplify
   Along the way, can throw out terms to simplify, if this is an upper-bound O() calculation.

Example: Sum of Integer Queue

```java
sum_queue(Q) {
    if (Q.length == 0) return 0;
    else return Q.dequeue() + sum_queue(Q);
}
```

- One subproblem
- Linear reduction in size (decrease by 1)
- Combining: constant c(s), 1 subproblem

Equation:

\[
T(0) \leq b \\
T(n) \leq c + T(n-1) \quad \text{for } n>0
\]
Example: Binary Search

Example: MergeSort

Lower Bound Analysis: Recursive Fibonacci

Analysis

Learning from Analysis

Logs and exponents
Logs and exponents

- We will be dealing mostly with binary numbers (base 2)
- Definition: \( \log_2 B = A \) means \( X^A = B \)
- Any base is equivalent to base 2 within a constant factor:
  \[ \log_j B = \frac{\log_j B}{\log_j X} \]

Why?

Because: if \( R = \log_2 B, S = \log_2 X, \) and \( T = \log_3 B, \)
- \( 2^R = B, 2^S = X, \) and \( X^T = B \)
- \( 2^R = X^T = 2^{ST} \) i.e. \( R = ST \) and therefore, \( T = R/S \).

Properties of logs

- We will assume logs to base 2 unless specified otherwise
- \( \log AB = \log A + \log B \) (note: \( \log AB \neq \log A \cdot \log B \))
- \( \log A/B = \log A - \log B \) (note: \( \log A/B \neq \log A / \log B \))
- \( \log A^K = K \log A \) (note: \( \log A^K \neq (\log A)^K \))
- \( \log \log X < \log X < X \) for all \( X > 0 \)
  - \( \log \log X = Y \) means \( 2^Y = X \)
  - \( \log X \) grows slower than \( X \); called a “sub-linear” function
- \( \log 1 = 0, \log 2 = 1, \log 1024 = 10 \)
Log-log plot

Kinds of Analysis
- So far we have considered worst case analysis
- We may want to know how an algorithm performs “on average”
- Several distinct senses of “on average”
  - amortized
    - average time per operation over a sequence of operations
  - average case
    - average time over a random distribution of inputs
  - expected case
    - average time for a randomized algorithm over different random seeds for any input

Amortized Analysis
- Consider any sequence of operations applied to a data structure
  - your worst enemy could choose the sequence?
- Some operations may be fast, others slow
- Goal: show that the average time per operation is still good
  \[
  \text{total time for } n \text{ operations} \div n
  \]

Stack ADT
- Stack operations
  - push
  - pop
  - isEmpty
- Stack property: if x is on the stack before y is pushed, then x will be popped after y is popped
  What is biggest problem with an array implementation?

Stretchy Stack Implementation
```
int * data;
int maxSize;
int top;

Push(e)
if (top == maxSize)
    temp = new int[2*maxSize];
    for (i=0;i<maxSize;i++) temp[i] = data[i];
    delete data;
    data = temp;
    maxSize = 2*maxSize;
else { data[++top] = e; }
```

Stretchy Stack Amortized Analysis
- Consider sequence of n operations
  - push(3); push(19); push(2); ...
- What is the max number of stretches?
- What is the total time?
  - let’s say a regular push takes time a, and stretching an array contain k elements takes time bk.

- Amortized time =
Stretchy Stack Amortized Analysis

- Consider sequence of n operations:
push(3); push(19); push(2); ...
- What is the max number of stretches? \( \log n \)
- What is the total time?
  - Let’s say a regular push takes time \( a \), and stretching an array containing \( k \) elements takes time \( bk \).
  
  \[
  an + b(1 + 2 + 4 + 8 + ... + n) = an + b \sum_{i=0}^{n-1} 2^i
  
  = an + b(2n - 1)
  \]
- Amortized time =

Stretchy Stack Amortized Analysis

- Consider sequence of n operations:
push(3); push(19); push(2); ...
- What is the max number of stretches? \( \log n \)
- What is the total time?
  - Let’s say a regular push takes time \( a \), and stretching an array containing \( k \) elements takes time \( bk \).

\[
\text{Amortized time} = \frac{(an + b(2n - 1))}{n} = O(\log n)
\]

Series

- Arithmetic series:
  \[
  \sum_{i=0}^{n} \frac{N(N+1)}{2}
  \]
- Geometric series:
  \[
  \sum_{i=0}^{n} \frac{A^n - 1}{A - 1}
  \]

\[
\sum_{i=0}^{\log n} 2^i = \frac{2^{\log n + 1} - 1}{2 - 1} = 2^{\log n} - 1 = 2n - 1
\]

Moral of the Story

To Do

- Assignment #1 due:
  - Electronic turnin: midnight, Monday Jan 21
  - Hardcopy writeup due in class Wednesday, Jan 23

- Finish reading Chapter 3,
  - Be prepared to discuss these questions (bring written notes to refer to):
    1. What is a call stack?
    2. Could you write a compiler that did not use one?
    3. What data structure does a printer queue use?