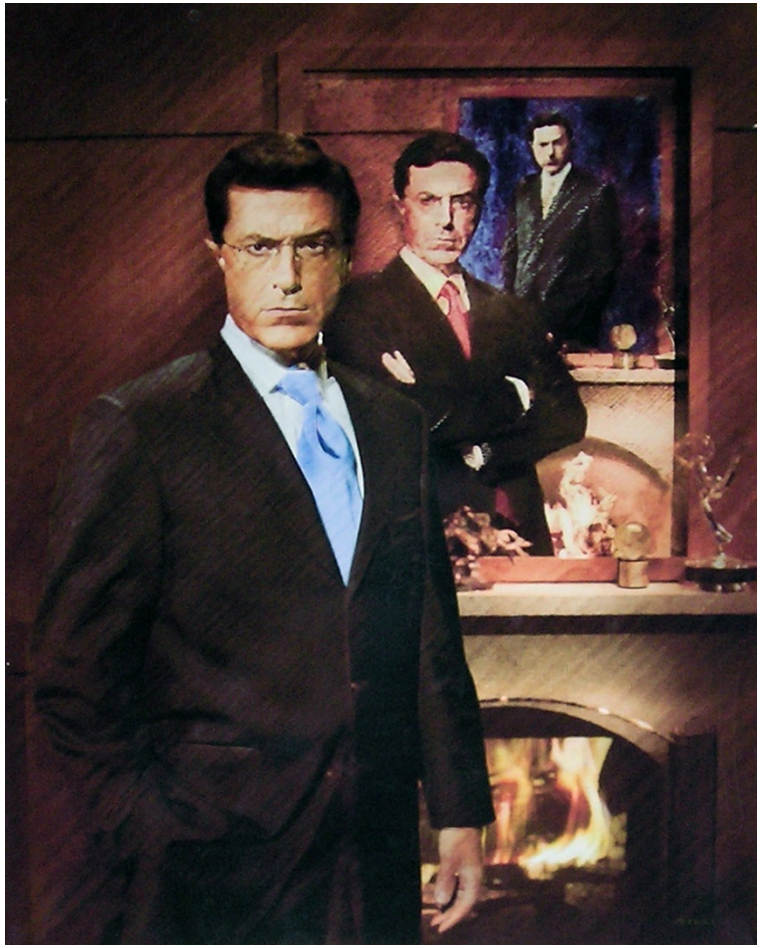


CSSE 220 Day 14

Sierpiński, Recursion and Efficiency, Mutual Recursion

Checkout *Recursion2* and *SortingAndSearching* projects from SVN

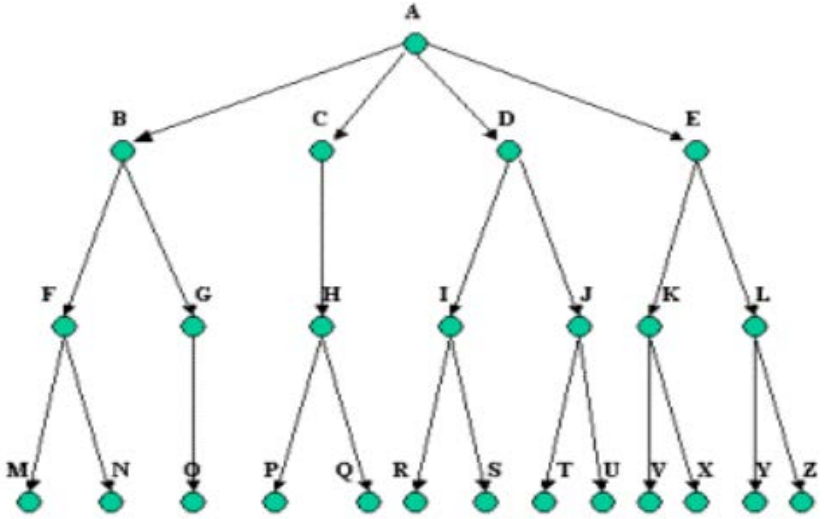

Bizarro
I have this recurring dream that I'm lying here telling you about a recurring dream about lying here telling you about a recurring dream about...
zzzz
©DAN PIRARO
3·6·02
BIZARRO.com
Dist. by Universal Press Synd.

Recap: What are recursive methods?

- Any method that calls itself
  - On a simpler problem
  - So that it makes progress toward completion
  - **Indirect recursion**: May call another method which calls back to it.

When should recursive methods be used?

- When implementing a recursive definition
- When implementing methods on recursive data structures



- Where parts of the whole look like smaller versions of the whole

Q1

The pros and cons of recursive methods

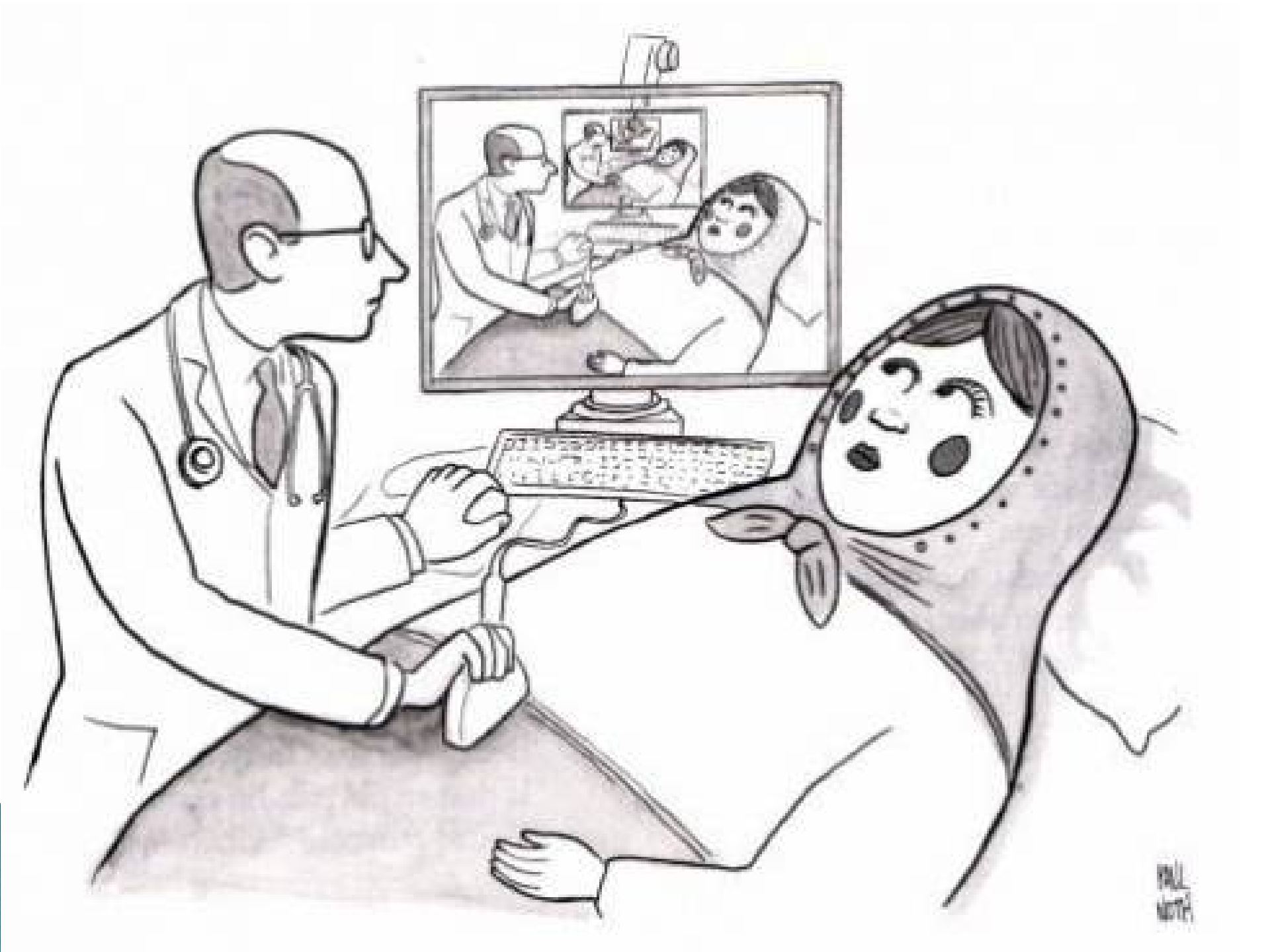
- The pros
  - easy to implement,
  - easy to understand code,
  - easy to prove code correct

- The cons
  - Sometimes takes more space and time than equivalent iterative solution
  - Why?
    - because of function calls

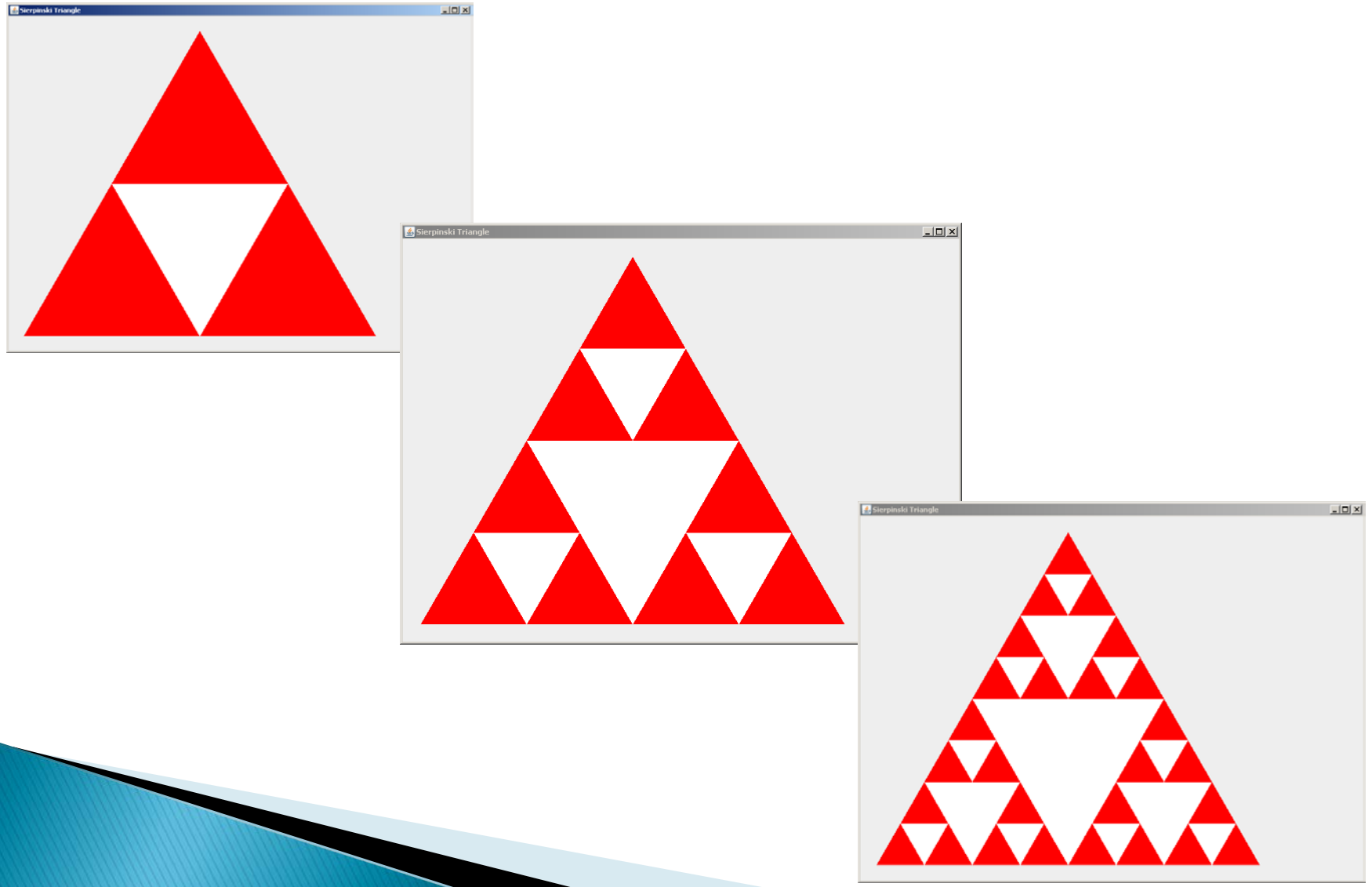
Q2

Recap: Key Rules to Using Recursion

- Always have a **base case** that **doesn't recurse**
- Make sure recursive case always makes **progress**, by **solving a smaller problem**
- **You gotta believe**
  - Trust in the recursive solution
  - Just consider one step at a time



HW: Sierpinski



Can one little Fib hurt?

- Why does recursive Fibonacci take so long?!?
- Can we fix it?

```
private static long fib(int n) {
    // TODO: Convert this to use memoization.
    long f;
    if (n <= 2) {
        f = 1;
    } else {
        long fNMOne = fib(n - 1);
        long fNMTwo = fib(n - 2);
        f = fNMOne + fNMTwo;
    }
    return f;
}
```

Q3

Memoization

- Save every solution we find to sub-problems
- Before recursively computing a solution:
  - Look it up
  - If found, use it
  - Otherwise do the recursive computation

Q4

Classic Time-Space Trade Off

- A deep discovery of computer science
- In a wide variety of problems we can tune the solution by varying the amount of storage space used and the amount of computation performed
- Studied by “Complexity Theorists”
- Used everyday by software engineers

Mutual Recursion

- 2 or more methods call each other repeatedly
  - E.g., Hofstadter Female and Male Sequences

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ n - M(F(n-1)) & \text{if } n > 0 \end{cases}$$

$$M(n) = \begin{cases} 0 & \text{if } n = 0 \\ n - F(M(n-1)) & \text{if } n > 0 \end{cases}$$

  - In how many positions do the sequences differ among the first 50 positions? first 500? first 5,000? first 5,000,000?

http://en.wikipedia.org/wiki/Hofstadter_sequence

Q5

```java
public int factorialRecursively(int n) {
    if (n <= 1) {
        return 1;
    }
    // Simple recursive implementation:
    int factNMinusOne = this.factorialRecursively(n - 1);
    return n * factNMinusOne;
}
```

```java
private static long fib(int n) {
    // TODO: Convert this to use memoization.
    long f;
    if (n <= 2) {
        f = 1;
    } else {
        long fNMOne = fib(n - 1);
        long fNMTwo = fib(n - 2);
        f = fNMOne + fNMTwo;
    }
    return f;
}
```

```java
private static void showMoves(int height, int start, int end, int temp) {
    if (height <= 0) {
        return;
    }

    showMoves(height - 1, start, temp, end);
    System.out.println("Move from " + start + " to " + end);
    TowersOfHanoi.count++;
    showMoves(height - 1, temp, end, start);
}
```

```java
private void drawSierpinski(Graphics2D g, double left, double
        double base) {
    if (base < 5) return;
```

```java
    double newBase = base/2.0;
    drawSierpinski(g, p1.getX(), p1.getY(), newBase);
    drawSierpinski(g, mp12.getX(), mp12.getY(), newBase);
    drawSierpinski(g, mp31.getX(), mp31.getY(), newBase);
}
```

Recursion Recap of 3 Rules

What is sorting?

- Let's see…

Why study sorting?

- Shlemiel the Painter

Shlemiel gets a job as a street painter, painting the dotted lines down the middle of the road. On the first day he takes a can of paint out to the road and finishes 300 yards of the road. "That's pretty good!" says his boss, "you're a fast worker!" and pays him a kopeck.

The next day Shlemiel only gets 150 yards done. "Well, that's not nearly as good as yesterday, but you're still a fast worker. 150 yards is respectable," and pays him a kopeck.

The next day Shlemiel paints 30 yards of the road. "Only 30!" shouts his boss. "That's unacceptable! On the first day you did ten times that much work! What's going on?"

"I can't help it," says Shlemiel. "Every day I get farther and farther away from the paint can!"

Course Goals for Sorting: You should…

- Be able to describe basic sorting algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
- Know the run-time efficiency of each
- Know the best and worst case inputs for each

Selection Sort

- Basic idea:
  - Think of the list as having a sorted part (at the beginning) and an unsorted part (the rest)
  - Find the smallest value in the unsorted part
  - Move it to the end of the sorted part (making the sorted part bigger and the unsorted part smaller)

Repeat until unsorted part is empty

Profiling Selection Sort

- Profiling: collecting data on the run-time behavior of an algorithm

- How long does selection sort take on:
  - 10,000 elements?
  - 20,000 elements?
  - …
  - 80,000 elements?

Q6

Analyzing Selection Sort

- Analyzing: calculating the performance of an algorithm by studying how it works, typically mathematically
- Typically we want the relative performance as a function of input size

- Example: For an array of length *n*, how many times does **`selectionSort()`** call **`compareTo()`**?

Handy Fact
$1 + 2 + \ldots + (n-1) + n = \frac{n(n+1)}{2}$

Q7–Q12

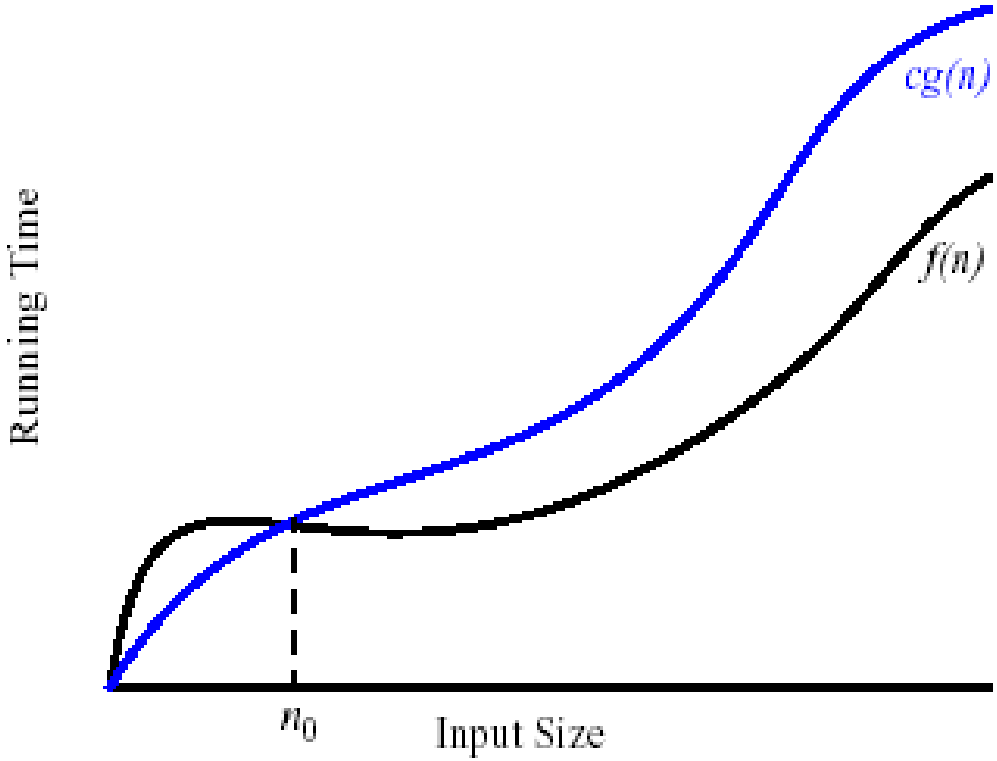
Big-Oh Notation

- In analysis of algorithms we care about differences between algorithms on very large inputs
- We say, “selection sort takes on the order of n^2 steps”
- Big-Oh gives a formal definition for “on the order of”

Q13

Formally

- We write $f(n) = O(g(n))$, and say "f is big-Oh of g"
- if there exists positive constants c and n_0 such that
- $0 \leq f(n) \leq c\ g(n)$ for all $n > n_0$
- g is a ceiling on f



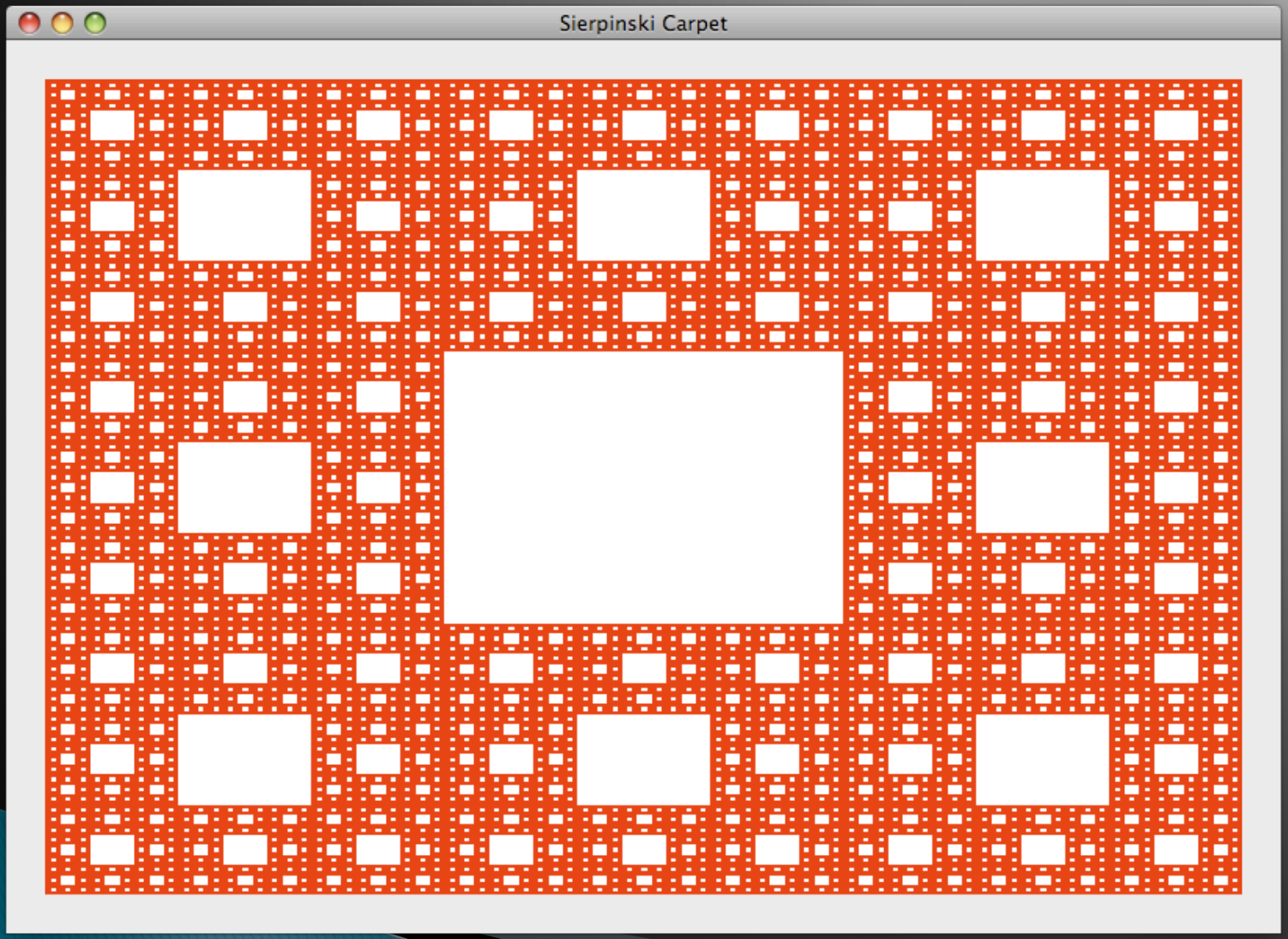
LodeRunner Teams

csse220-201330-LR01,abeggleg,araujol,greenwpd
csse220-201330-LR02,benshorm,mcnelljd,woodjl
csse220-201330-LR03,daruwakj,holzmajj,kadelatj
csse220-201330-LR04,gauvrepd,hazzargm,songh1
csse220-201330-LR05,gouldsa,malikjp,olivernp
csse220-201330-LR06,griffibp,heathpr,tebbeam
csse220-201330-LR07,litwinsh,plugerar,shumatdp
csse220-201330-LR08,adamoam,alayonkj,vanakema
csse220-201330-LR09,bochnoej,johnsotb,tatejl
csse220-201330-LR10,calhouaj,cheungnj,walthecn
csse220-201330-LR11,evansc,wagnercj,roccoma

Cont.

csse220-201330-LR12,haloskzd,mookher,stephaje
csse220-201330-LR13,hullzr,naylorbl,winterc1
csse220-201330-LR14,johnsoaa,kethirs,wrightj3
csse220-201330-LR15,liuj1,phillics,zhoup

Homework: Sierpinski Carpet



Q14-15