FALL 2003 CS211
SECTION 3

❒ Announcements
- many section files posted
- applications.html
- lecture files on the way!
- A2 due next week

❒ Overview
- answer questions
- recursion
- tail recursion
- Towers of Hanoi

1. Recursion

1.1 Induction

- show that induction process helps to "wire" your brain for recursion
- if you can identify base case, inductive hypothesis, and inductive step, you're very close!

1.2 Example

- iterative sum of n integers :
  - S(0) = 0
  - S(1) = 1 + 0 = 1
  - S(2) = 2 + 1 + 0 = 3
  - S(3) = 3 + 2 + 1 = 6
  - …
  - S(n) = (n+1)*n/2

- recursive sum of n integers:
  - S(0) = 0
  - S(n) = n + S(n-1)
  - check:
    S(1) = 1 + S(0) = 1
    S(2) = 2 + S(1) = 2 + 1 + 0 = 3
    S(3) = 3 + S(2) = 3 + 2 + 1 + 0 = 6
- identical, but completely different ways to state
  - iterative screams of loops
  - recursive is … well, recursive

1.3 Iterative Solution

- Algorithm:
  - get n >= 0
  - count ← 0, sum ← 0
  - if count <= n, sum increments by count
  - repeat
- Code:

```
public class IterativeSum {
    public static void main(String[] args) {
        final int N = Integer.parseInt(args[0]);
        int sum = 0;
        for (int k = 0 ; k <= N ; sum+=k, k++);
        System.out.println(sum);
    }
}
```

1.4 Recursive Solution

- Algorithm:
  - get n.
  - if n is 0, sum ← 0
  - otherwise, sum ← n + sum(n-1)
- Code:

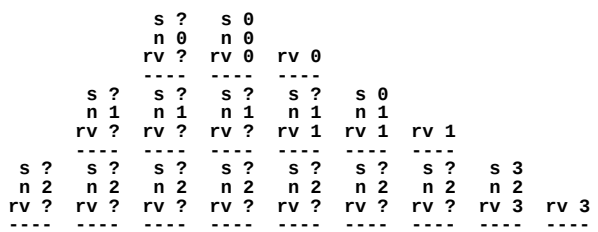
```
public class RecursiveSum {

    public static void main(String[] args) {

        final int N = Integer.parseInt(args[0]);
        int sum = sum(N);
        System.out.println(sum);

    }

    private static int sum(int n) {
        if (n==0)
            return 0;
        else
            return n + sum(n-1);
        }
    }

} // Class RecursiveSum
```


1.5 Alternative Recursive Solution

```
public class RecursiveSumAlt {

    public static void main(String[] args) {

        final int N = Integer.parseInt(args[0]);
        int sum = sum(N);
        System.out.println(sum);

    }

    private static int sum(int n) {
        int sum;
        if (n==0)
            sum = 0;
        else
            sum = n + sum(n-1);
        return sum;
    }

} // Class RecursiveSumAlt
```


1.6 Cool Concepts

- Computational path for a recursive series is *two-way*:
  - 1st path goes up: recursive calls pile up on stack
  - 2nd path goes down: answers combined together
  - So, the 1st path breaks the problem down into simple basic components, and the 2nd path assembles the sol
- You can calculate really complex things using recursion with simple sub-processes.
- Two essential parts of any recursive definition:
  - Base case(s): tells the recursion when to stop
  - Recursive step: tells the recursion how to break a problem into an operation it knows (e.g., addition) and a simpler problem (S(n-1))
- sum(2) example:



2. Tail Recursion

2.1 Definition

- ***tail recursion***: last action by recursive method is a recursive call
- generally can easily convert tail recursive method into an iterative (loop) form
- see example from before: I counted how many sums I needed to know

2.2 Why?

- recursion builds frame upon frame on the stack
- consumes large amount of memory if recursion is deep
- space efficiency can be improved by jumping up and down in the *same* frame for one method call
- will see this issue later in asymptotic complexity