- Project 3: code reviews due today
- Project 4: due Wednesday
- Project 5: discussion

Code Craft

Recursion

- Elephants…

Solving Big Problems

How do we solve big problems in CS?

Solving Big Problems

How do we solve big problems in CS?

- Break into smaller problems and solve each of these
- Then, put the pieces back together

In many cases, the structure of these smaller problems looks a lot like that of the original one

Self-Similarity

This self-similar structure suggests a certain type of tool: recursion

- Re-use same code at each level
- This means that we will be writing methods that call themselves!

Implementing Recursive Methods

What pieces of code do we need?

Implementing Recursive Methods

What pieces of code do we need?

- How to break up a big problem into smaller pieces
- How to put each sub-solution back together
- Recognizing when the problem is small enough that we can solve easily in code (without a recursive call)

Implementing Recursive Methods

Within the method, you have two essential cases:

- ***Base case***: small problem that we solve directly
  - This must be well defined!
- ***Recursive case***: break into smaller problems and call recursive method on these

Before you implement, identify these cases!

Example: Evaluating Expressions

- …

Example: Building Fractals

Example: N choose K

Notes

- Method calls require a certain amount of "overhead"
  - Time and memory
- Any algorithm that is implemented as a loop can also be implemented recursively
  - Would we want to do this?

Notes

How about the other way around: can any recursive algorithm be implemented with a loop?

Notes

How about the other way around: can any recursive algorithm be implemented with a loop?

- Yes: but you would also need a stack data structure to keep track of all of the work left to do
- Note that method calls ***are*** stack operations

Efficiency Choices

Loop versus recursion

- The choice comes down to your specific situation
- In general:
  - Loops are more efficient with respect to time and memory, but need more work
  - Recursion is often more elegant, but can cost time and memory

Other Recursion Examples

- Find all files in a directory whose name contains a specific sequence of characters
- Towers of Hanoi
- String parsing
- Parsing programs (compilers)
- Sorting
  - Quicksort
  - Mergesort

Example Merge Sort

Example Merge Sort

- Cut array into two halves
- Sort each half (recursive call)
- Merge the two arrays back together

What is the base case?

Merge Sort: Analysis

- What is the best case for merge sort?
- What is the worst case?
- How well does this algorithm do with data that is already sorted (completely or partially)?

Merge sort is one of the most efficient implementations of sort

Merge Sort: Analysis

Fundamental problem:

- We need a lot of extra memory (equal to the number of elements to be sorted)

Quicksort

- Perform operations within the original array
- Only ~ log2(n) of extra space

Quicksort

Sorting a sub-array between indices i and j

- Pick one of the values to be the ***pivot***
  - Could be x[i] or could be x[(i+j)/2]
- As long as x[i] <= pivot, increment i
- As long as x[j] >= pivot, decrement j
- Swap elements i & j
- Repeat until i & j cross
- Now have two sub-arrays i … c and c+1 … j
- Sort each separately

Quicksort Analysis

- What is the best case for quicksort?
- What is the worst case?
- How well does this algorithm do with data that is already sorted (completely or partially)?

Quicksort Analysis

- Average case: better than merge sort
- Worst case: worse than merge sort

Recursion Wrap-Up

- Recursion is a key tool in CS
  - Requires practice
- It isn't a tool for every problem
  - Often a loop will do the job. In these cases, use the loop
- If you do recursion: don't forget the base cases!