RECURSION

CMPS 148

Recursion

- Recursion refers to whenever a function calls *itself* - either directly or indirectly
- No problem *requires* recursion - but some problems are ***very*** difficult without it.
- General Rule: When ever you can solve a problem by breaking it into smaller problems (and then re-assembling), recursion c*ould* work…

Recursion

- Steps:
  - Identify base case (simple case)
  - Identify method of breaking up problem

- Example 1: Calculate N!

Fibonacci Numbers

- Fibonacci Series is a sequence of numbers where each number is the sum of the last 2:
  - Fib(0) = 1
  - Fib(1) = 1
  - Fib(2) = Fib(1) + Fib(0) = 2
  - Fib(3) = Fib(2) + Fib(1) = 3
  - Fib(4) = Fib(3) + Fib(2) = 5
  - Fib(5) = Fib(4) + Fib(3) = 8
  - Fib(6) = Fib(5) + Fib(4) = 13….

Recursion: Pitfalls

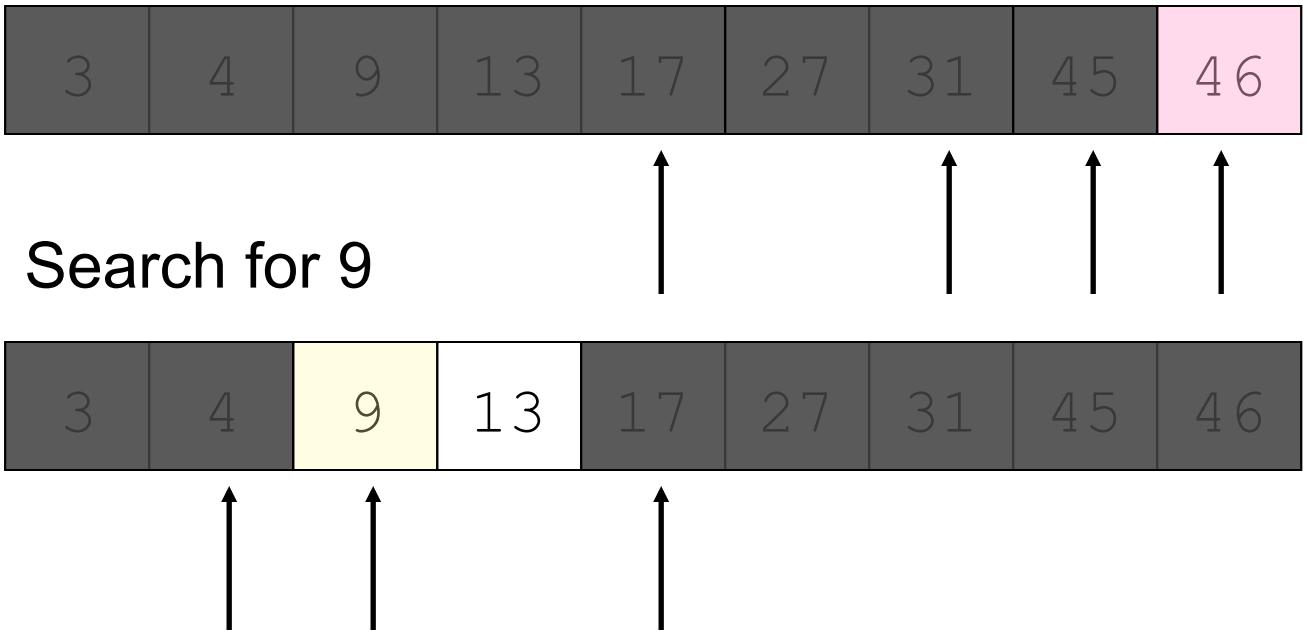
- Although it looks like a good solution, calculating factorials and Fibonacci numbers recursively is **very** expensive
  - Function calls take time and **memory**
    - **Parameters and Return location are placed on *call stack***
  - Calculating factorial of 10 requires 10 functions calls 1000! requires 1000… not good!
  - Typically, we use recursion when we can break the problem (roughly) in half.

Binary Search

- Given an ordered list, binary search can find an item in logN time!
  - Algorithm:
    - Look at middle item. If it's the value, you're done.
    - If too large, discard everything "above". If too small, discard everything "below"
    - Repeat while you have more than 2 items
    - ***Guaranteed to find within $\log_2 N$***

Binary Search

- Real Example: Search for 46



Recursive Binary Search

- Search List:
  - Key, Array
  - Start (start index in array)
  - End (end index in array)
- Procedure:
  - Find mid point. If not key, either
    - Search with start = mid+1, end = end
    - Or Search with start = start and end = mid – 1

```
search (int [] values, int start, int end, int key)
```

Lab 4 (see appiversity)

Euclid's algorithm.

The *greatest common divisor* (gcd) of two positive integers is the largest integer that divides evenly into both of them. For example, the greatest common divisor of 102 and 68 is 34 since both 102 and 68 are multiples of 34, but no integer larger than 34 divides evenly into 102 and 68.

We can efficiently compute the gcd using the following property, which holds for positive integers p and q:

If $p > q$, the gcd of p and q is the same as the gcd of q and $p \% q$.

Write a ***recursive gcd function*** (and a main function to test it) that uses this algorithm.