CSCI 136
Data Structures &
Advanced Programming

Lecture 7

Fall 2019

Instructors: B&S

Last Time

- Vector Implementation
- Miscellany: Wrappers
- Condition Checking
  - Pre- and post-conditions, Assertions

Today

- Asymptotic Growth & Measuring Complexity (from previous slide deck)
- Introduction to Recursion & Induction

Recursion

- General problem solving strategy
  - Break problem into smaller pieces (sub-problems)
  - Sub-problems are typically smaller versions of same problem

Recursion

- Many algorithms are recursive
  - Can be easier to understand, prove correctness, or determine efficiency
- Today we will review recursion and then talk about techniques for reasoning about recursive algorithms

Factorial

- n! = n • (n-1) • (n-2) • … • 1
- How can we implement this?
  - We could use a for loop…

```
int product = 1;
for(int i = 1;i <= n; i++)
      product *= i;
```

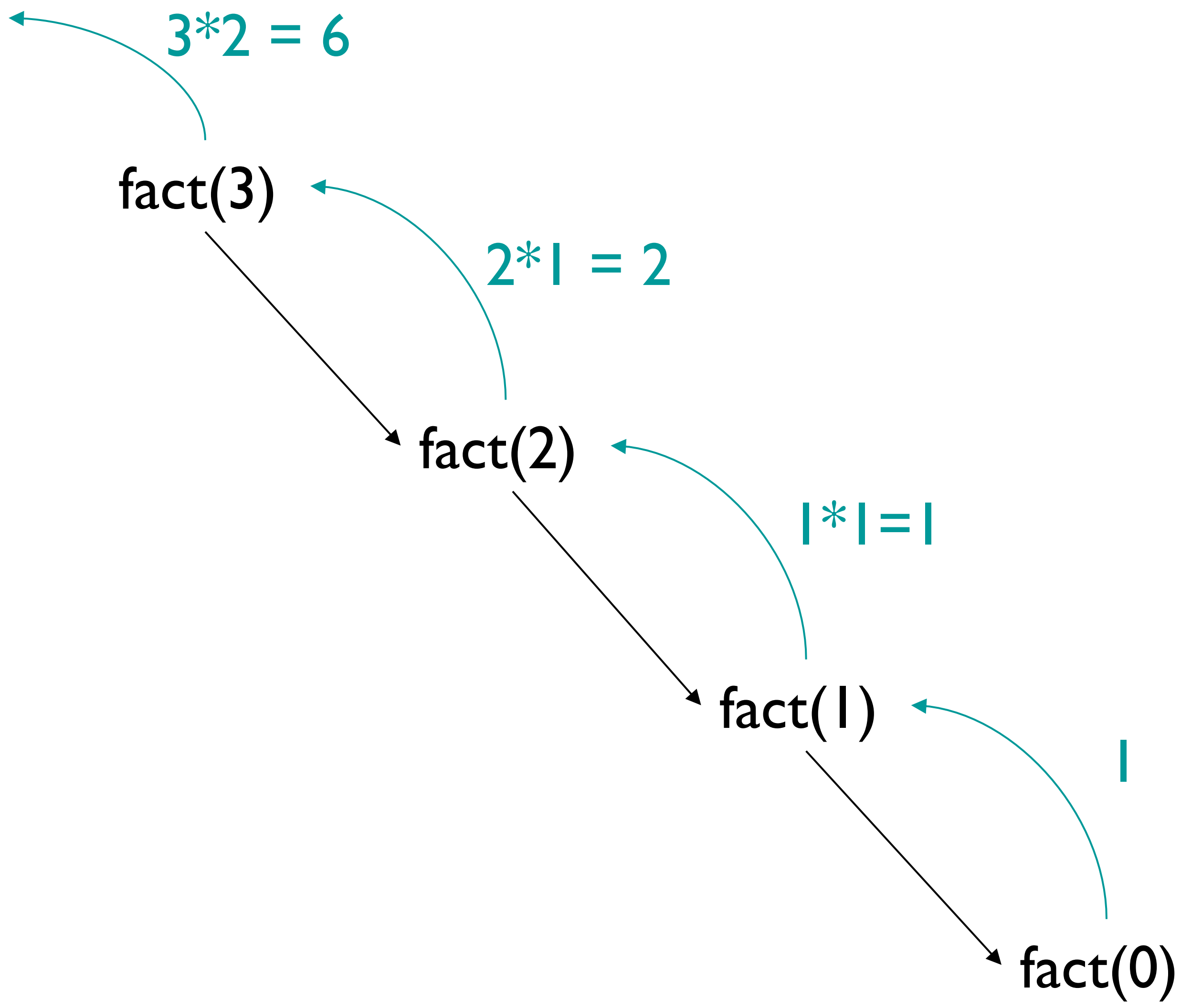
- But we could also write it recursively….

Factorial

- $n! = n \bullet (n-1) \bullet (n-2) \bullet \ldots \bullet 1$
- Recursive definition (what "…" really means!)
  - $n! = n \bullet (n-1)!$
  - $0! = 1$

```
// Pre: n >= 0
public static int fact(int n) {
      if (n==0) return 1;
      else return n*fact(n-1);
}
```

Factorial



Factorial

- In recursion, we always use the same basic approach
- What's our base case? [Sometimes "cases"]
  - n=0; fact(0) = 1
- What's the recursive relationship?
  - n>0; fact(n) = n • fact(n-1)

Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, ...
- Definition
  - $F_0 = 1$, $F_1 = 1$
  - For $n > 1$, $F_n = F_{n-1} + F_{n-2}$
- Inherently recursive!
- It appears almost everywhere
  - Growth: Populations, plant features
  - Architecture
  - Data Structures!

fib.java

```
public class fib{
  // pre: n is non-negative
   public static int fib(int n) {
      if (n==0 || n == 1) {
         return 1;
      }
      else {
         return fib(n – 1) + fib(n – 2);
      }
   }

   public static void main(String args[]) {
      System.out.println(fib(Integer.valueOf(args[0]).intValue()));
   }

}
```

Demo: RecursiveMethods.java….

Question: Why is fib so slow?!

Towers of Hanoi

- Demo
- Base case:
  - One disk: Move from start to finish
- Recursive case (n disks):
  - Move smallest n-1 disks from start to temp
  - Move bottom disk from start to finish
  - Move smallest n-1 disks from temp to finish
- Let’s try to write it....

Recursion Tradeoffs

- Advantages
  - Often easier to construct recursive solution
  - Code is usually cleaner
  - Some problems do not have obvious non-recursive solutions
- Disadvantages
  - Overhead of recursive calls
  - Can use lots of memory (need to store state for each recursive call until base case is reached)
    - E.g. recursive fibonacci method

Alternate contains() for Vector

```
// Helper method: returns true if elt has index in range from..to
public boolean contains(E elt, int from, int to) {
   if (from > to)
        return false; // Base case: empty range
   else
        return elt.equals(elementData[from]) ||
               contains(elt, from+1, to);
}

public boolean contains(E elt) {
   return contains(elt, 0, size()-1); }
```

- What's the time complexity of contains?
  - O(to – from + 1) = O(n) (n is the portion of the array searched)
  - Why?
    - Bootstrapping argument! True for: to – from = 0, to – from = 1, …
- Let's formalize this bootstrapping idea....

Mathematical Induction

- The mathematical cousin of recursion is induction
- Induction is a proof technique
- Reflects the structure of the natural numbers
- Use to simultaneously prove an infinite number of theorems!

Mathematical Induction

- Example: Prove that for every n ≥ 0

$$P_n : \sum_{i=0}^{n} i = 0 + 1 + \ \dots + n = \frac{n(n+1)}{2}$$

- Proof by induction:
  - Base case: P_n is true for n = 0 (just check it!)
  - Induction step: If P_n is true for some n≥0, then P_{n+1} is true.

$$P_{n+1}: 0 + 1 + \ \dots + n + (n+1) = \frac{(n+1)\big((n+1)+1\big)}{2} = \frac{(n+1)(n+2)}{2}$$

Check: $0 + 1 + \ \dots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2}$

- First equality holds by assumed truth of P_n!

Mathematical Induction

Principle of Mathematical Induction (Weak)

Let P(0), P(1), P(2), ... Be a sequence of statements, each of which could be either true or false. Suppose that

1. P(0) is true, and
2. For all $n \geq 0$, if P(n) is true, then so is P(n+1).

Then all of the statements are true!

Note: Often Property 2 is stated as

2. For all $n > 0$, if P(n-1) is true, then so is P(n).

   Apology: I do this a lot, as you'll see on future slides!

Mathematical Induction

- Prove: $\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$

- Prove: $0^3 + 1^3 + \ldots + n^3 = (0 + 1 + \ldots + n)^2$

Proof: $0^3 + 1^3 + \ldots + n^3 = (0 + 1 + \ldots + n)^2$

Note: I'm doing the n-1 → n version

$$
\begin{aligned}
(0^3 + 1^3 + \ldots n^3) &= (0^3 + 1^3 + \ldots + (n-1)^3) + n^3 \\
\text{Induction☞} \quad &= (0 + 1 + \ldots + (n-1))^2 + n^3 \\
&= \left(\frac{n(n-1)}{2}\right)^2 + n^3 \\
&= n^2 \left(\frac{(n-1)^2 + 4n}{4}\right) \\
&= n^2 \left(\frac{n^2 + 2n + 1}{4}\right) \\
&= n^2 \left(\frac{(n+1)^2}{4}\right) \\
&= \left(\frac{n(n+1)}{2}\right)^2 \\
&= (0 + 1 + \ldots + n)^2
\end{aligned}
$$

What about Recursion?

- What does induction have to do with recursion?
  - Same form!
    - Base case
    - Inductive case that uses simpler form of problem
- Example: factorial
  - Prove that fact(n) requires n multiplications
    - Base case: n = 0 returns 1, 0 multiplications
    - Assume true for all k<n, so fact(k) requires k multiplications.
    - fact(n) performs one multiplication (n*fact(n-1)). We know that fact(n-1) requires n-1 multiplications. 1+n-1 = n, therefore fact(n) requires n multiplications.

Counting Method Calls

- Example: Fibonacci
  - Prove that fib(n) makes at least fib(n) calls to fib()
    - Base cases: n = 0: 1 call; n = 1; 1 call
    - Assume that for some n ≥ 2, fib(n-1) makes at least n-1 calls to fib() and fib(n-2) makes at least fib(n-2) calls to fib().
    - Claim: Then fib(n) makes at least fib(n) calls to fib()
      - 1 initial call: fib(n)
      - By induction: At least fib(n-1) calls for fib(n-1)
      - And as least fib(n-2) calls for fib(n-2)
      - Total: 1 + fib(n-1) + fib(n-2) > fib(n-1) + fib(n-2) = fib(n) calls
    - Note: Need two base cases!
  - One can show by induction that for n > 10: fib(n) > $(1.5)^n$
  - Thus the number of calls grows exponentially!
  - We can visualize this with a *method call graph*….

Mathematical Induction : Version 2

Principle of Mathematical Induction (Weak)

Let P_0, P_1, P_2, ... Be a sequence of statements, each of which could be either true or false. Suppose that

1. P_0 and P_1 are true, and
2. For all $n \geq 2$, if P_{n-1} and P_{n-2} are true, then so is P_n.

Then all of the statements are true!

Other versions:

- Can have $k > 2$ base cases
- Doesn't need to start at 0

Example: Binary Search

- Given an array a[] of positive integers in increasing order, and an integer x, find location of x in a[].
  - Take "indexOf" approach: return -1 if x is not in a[]

```
protected static int recBinarySearch(int a[], int value,
             int low, int high) {
  if (low > high) return -1;
  else {
      int mid = (low + high) / 2;        //find midpoint
      if (a[mid] == value) return mid;  //first comparison
                                         //second comparison
      else if (a[mid] < value)           //search upper half
      return recBinarySearch(a, value, mid + 1, high);
       else                              //search lower half
            return recBinarySearch(a, value, low, mid - 1);
  }
```

Binary Search takes O(log n) Time

Can we use induction to prove this?

- Claim: If n = high - low + 1, then recBinSearch performs at most c (1+ log n) operations, where c is *twice* the number of statements in recBinSearch
- Base case: n = 1: Then low = high so only c statements execute (method runs twice) and c **≤** c(1+log 1)
- Assume that claim holds for some n ≥ 1, does it hold for n+1? [Note: n+1 > 1, so low < high]
- Problem: Recursive call is *not* on n : it's on n/2.
- Solution: We need a better version of the PMI….

Mathematical Induction

Principle of Mathematical Induction (Strong)

Let P(0), P(1), P(2), ... Be a sequence of statements, each of which could be either true or false. Suppose that, for some $k \geq 0$

1. P(0), P(1), ... , P(k) are true, and
2. For every $n \geq k$, if P(1), P(2), ... , P(n) are true, then so is P(n+1).

Then all of the statements are true!

Binary Search takes O(log n) Time

Try again now:

- Assume that for some $n \geq 1$, the claim holds *for all* $k \geq n$, does claim hold for $n+1$?
- Yes! Either
  - $x = a[mid]$, so a constant number of operations are performed, or
  - RecBinSearch is called on a sub-array of size $n/2$, and by induction, at most $c(1 + \log(n/2))$ operations are performed.
    - This gives a total of at most $c + c(1 + \log(n/2)) = c + c(\log(2) + \log(n/2)) = c + c(\log n) = c(1 + \log n)$ statements

Wait…what??? (think about this as you go to sleep)

- Prove: All horses are the same color.
- Base case: n = 1. Clear
- Induction (n>1): Assume we have a set X of n horses. Let x and y be two of the horses. X – {x} is a set of n-1 horses, so (by induction) they are all the same color. Similarly, all horses in X – {y} are the same color. Now pick z in X, z $\neq$ x,y. Then z is in X-{x} and z is in X-{y}, so all all horses are the same color (as z)!
- Question: What went wrong?