Recursion

Recursive method is a method that calls itself

Could lead to simple ways of solving iterative problems … in general if you have a for loop (or other iterative loop), you can usually solve the same problem with recursion (and vice versa)

However, there are certain problems that are more natural for recursion

* Problems that contain themselves as a subproblem

The best way to think about recursion is in terms of mathematical induction

* base case
* inductive case

Let’s look at one of the examples from last time again: factorial()

The factorial of a number, x!, is given by

1\*2\*3\*…\*(x-1)\*x

e.g. 5! = 1\*2\*3\*4\*5 = 120

* base case: x = 1, x! = 1! = 1
* recursive case: given (x-1)!, how do I calculate x!? … x! = x \* (x-1)!

Here is our recursive factorial method:

public int factorial(int x) {

if (x == 1)

return 1;

return x \* factorial(x-1);

}

How does this work:

x = 1: factorial(1) = 1;

x = 2: factorial(2) = 2\*factorial(1)

= 2\*1 = 2

x = 3: factorial(3) = 3\*factorial(2)

= 3\*2\*factorial(1)

= 3\*2\*1 = 6

x = 4: factorial(4) = 4\*factorial(3)

= 4\*3\*factorial(2)

= 4\*3\*2\*factorial(1)

= 4\*3\*2\*1 = 24

x = 5: factorial(5) = 5\*factorial(4)

= 5\*4\*factorial(3)

= 5\*4\*3\*factorial(2)

=5\*4\*3\*2\*factorial(1)

= 5\*4\*3\*2\*1 = 120

We are going to prove that:

* a given recursive method returns the right answer; and
* a given recursive method halts

To prove correctness:

* Prove that the base case (stopping case) returns the right answer
* For each recursive case, show that it returns the right answer if the recursive call returns the right answer
* Exactly like induction

1. Factorial method

* Prove that the base case is correct
  + factorial(1)
    - in this case x==1
    - “if” statement is true
    - return 1
    - this is the correct answer
* Prove that the recursive case is correct assuming that the recursive call gives the right answer
  + factorial(n) for n > 1
    - in this case x != 1
    - if statement is false – we do not return 1
    - go to the next line:

return n\*factorial(n-1);

Assuming factorial(n-1) returns the right answer, we know that n! = n\*(n-1)!

Therefore we return n!, which is correct

* + Finally, the base case is correct and the recursive case is correct assuming the recursive call returns the right answer, so the method is correct.

1. Prove that the following method returns 2^x (for x >= 0)

public int powerOf2(int x) {

// notice that 2^x = 2\*(2^(x-1))

if (x == 0)

return 1;

return 2\*powerOf2(x-1);

}

* The base case: x == 0
  + powerOf2(0)
    - x == 0 is true
    - if statement is true
    - return 1
    - 2^0 = 1 … correct
* The recursive case: x == n, n > 0
  + powerOf2(n), n > 0, assuming the recursive call returns the right answer
    - n != 0
    - if statement is false … we do not return 1
    - We do return 2\*powerOf2(n-1)
    - We assume

powerOf2(n-1) = 2^(n-1), the correct answer

When we return 2\*powerOf2(n-1) we are actually returning 2\*2^(n-1) = 2^n, which is the correct answer

* Base case and recursive case are both correct so the method is correct

Proving that a given recursive method halts

1. We define the size of each call of the recursive method. This size must be a nonnegative integer (“natural number”).
2. We prove that the size decreases each time the recursive method is called.

The base case must be at the smallest allowed size. These conditions guarantee that the recursive method will hit a stopping case, which means the method must halt.

e.g. Factorial method: factorial(x)

1. What is the size of factorial(x)? x

- x is an integer

- x is nonnegative – can’t call factorial on negative integers

2. Does x decrease with each recursive call?

- The recursive call looks like

x \* factorial(x-1)

- in the recursive call, we call factorial(x-1). The size of the current call is x, the size of the next call is x-1, which is strictly less than x.

The size is a nonnegative integer, and decreases with each recursive call, therefore it will eventually hit a stopping case, which means it will halt.

Prove that the powerOf2 method will halt.

powerOf2(x)

1. Size? x
   1. Is x an integer? Yes
   2. Is x nonnegative? Yes
2. Show that it decreases with each recursive call.
   1. Where is the recursive call?

2\*powerOf2(x-1)

When we call this method, we are inside the powerOf2(x) method with size x

We call powerOf2(x-1) with size x-1

x-1 is strictly less than x, therefore, the size decreases with each call

Therefore, powerOf2(x) will eventually hit a stopping case and halt