Algorithms & Complexity 4/3/17 - 4/5/17

0145-344-001

Note Taker: Jai Punjwani

ANNOUNCEMENTS

Topic: Recursion

PowerPoint: <http://home.adelphi.edu/~siegfried/cs344/344l9.pdf>

**Recursion** is when a method calls itself. The sequence of method calls stops with a base case.

Natural example: Fibonacci Sequence

Any iterative algorithm can be written recursively and vice-versa. The question is: **which one should we use? what are the tradeoffs?**

If the complexities of both approaches are the same, then opting for an iterative algorithm can save the method call stack, which can be expensive. Note that some problems have solutions that are naturally recursive and thus are readable.

Sometimes recursive methods are facilitated by **helper** methods, which allow you to define additional parameters to keep track of additional information.

ex: a method that prints a string backwards

public void printBackwards(String s) {

System.out.println(helper(s));

}

// helper method to reverse string

public String helper(String s) {

if (s.length() ==1)

return s;

return helper(s.substring(1)) + s.charAt(0); // places first character at end, and recursively calls //helper with the rest of the String

}

Note that we were easily able to do recursion and combine our results using the concatenation ‘+’ operator. This would have been trickier if we used print statements in our recursive call, which is why I opted for a helper in this case.

Challenge: Compute the sum of the first n Fibonacci numbers. Write this without having to compute the ith