**Practice Problems For Topic 6: Recursion**
*CISC 121, Fall 2014*

Your solutions for all of these practice problems must be recursive. You may not use any loops!

This list combines practice problems from three sources. The more you practice, the more comfortable you will be with recursion.

**Part A:** From the course notes about recursion, published on Moodle: do problems 1-4. Problem 5 is an example I plan to do in class. In problem 3, the writer refers to using math.random(), which is a Java method; these notes were originally written when we taught CISC 121 in Java. Use one of the functions in the Python `random` library instead.

**Part B:** These problems are from the text, but I'm copying them here to make sure that students with different versions/editions can find the problems I mean and so I can add some clarification.

**Exercise 2 from section 5.9:** Write a function called `do_n` that takes a function object and a number, `n`, as arguments, and that calls the given function `n` times.

*Margaret's note:* We haven't passed functions as parameters before, but this is a useful thing that Python allows. Here's a quick example:

```python
def greeting():
    print "hello, world!"

def twice(func):
    func()
    func()
    func()
```

If you call `twice(greeting)`, Python will print "hello, world!" two times.

**Exercise 7 from section 6.11:** A number, `a`, is a power of `b` if it is divisible by `b` and `a/b` is a power of `b`. Write a function called `is_power` that takes parameters `a` and `b` and returns `True` if `a` is a power of `b`. Note: you will have to think about the base case.

*(more on next page)*
Exercise 8 from section 6.11: The greatest common divisor (GCD) of \(a\) and \(b\) is the largest number that divides both of them with no remainder. One way to find the GCD of two numbers is Euclid’s algorithm, which is based on the observation that if \(r\) is the remainder when \(a\) is divided by \(b\), then \(\text{gcd}(a, b) = \text{gcd}(b, r)\). As a base case, we can use \(\text{gcd}(a, 0) = a\).

Write a function called \(\text{gcd}\) that takes parameters \(a\) and \(b\) and returns their greatest common divisor. If you need help, see \text{http://en.wikipedia.org/wiki/Euclidean_algorithm}.

Margaret’s note: The explanation from the book assumes that \(a \geq b\). If \(a < b\), just remember that \(\text{gcd}(a,b)\) equals \(\text{gcd}(b,a)\), so find the remainder of \(b/a\) instead of \(a/b\).

Here is an example of this algorithm in action:
\[
\text{gcd}(462, 165) = \text{gcd}(132, 165) \quad \text{since 462/165 has a remainder of 132}
\]
\[
= \text{gcd}(132, 33) \quad \text{since 165/132 has a remainder of 33}
\]
\[
= \text{gcd}(33,0) \quad \text{since 142/33 is exactly 4, with a remainder of 0}
\]
\[
= 33 \quad \text{by the base case}
\]

Your function may assume that neither of its parameters are negative.

Part 3:

1. Write a function called \(\text{findMin}\) that takes a list of numbers as a parameter and returns the minimum number from the list. You may assume that the list is not empty. You must do this \textit{without} using the Python \texttt{min} function.

2. Write a recursive version of binary search that takes a sorted list of numbers and a target value and returns the index of the target in the list (or \(-1\) if the value does not appear in the list). You must generalize binary search so that it can be called to sort just part of a list rather than the whole list. Your function should have four parameters:
   1. the list
   2. the target value
   3. lowest index to search
   4. highest index to search

You can specify default values for parameters 3 and 4 so that to search an entire list you can leave them out of your call. The default value for parameter 3 is obvious: zero. The default value for parameter 4 is a little more tricky; we’d like it to be the highest index in the list, but default parameter values must be constants. Here’s a little Python trick:

\[
\text{def binSearch(nums, target, lowIndex=0, highIndex=None):}
\]
\[
\quad \text{if highIndex == None:}
\]
\[
\quad \quad \text{highIndex = len(nums)-1}
\]
\[
\quad \# and continue....
\]

(example on the next page)
Here is an example of a use of this function:

```python
>>> nums=range(2,21,2)
>>> nums
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
>>> binSearch(nums,6)
2
>>> binSearch(nums,20)
9
>>> binSearch(nums,9)
-1
>>> binSearch(nums,30)
-1
>>> binSearch(nums,18,1,5)
-1
```