Adam Rich
EN.605.202.87.SP18 Data Structures
Module 2, Homework Assignment 3
February 15, 2018

Assignment 3 – Recursion

Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.

Q1 and Q2 are intended to help you get comfortable with recursion by thinking about something familiar in a recursive manner. Q3 – Q6 are practice in working with non-trivial recursive functions. Q7 and Q8 deal with the idea of conversion between iteration and recursion.

1. Write a recursive algorithm to compute *a+b*, where *a* and *b* are nonnegative integers.

```
method a_plus_b(int a, int b)
  if b == 1 return a + 1;
  return a_plus_b(a + 1, b – 1);
end-method
```

2. Let A be an array of integers. Write a recursive algorithm to compute the average of the elements of the array. Solutions calculating the sum recursively, instead of the average, are worth fewer points.

```
method average(int[] x, optional int[] weight)
    if weight is defined it has to have the same length as x
       else throw error

    if weight is not initialized
        set it to be a vector of 1s the same length as x

    if length of x is 1 return x[0]

    def n to be length of x (and weight since they are the same)
    def w to be weight[0] + weight[1]

    x[1] <- (x[0]*weight[0] + x[1]*weight[1]) / w
    weight[1] <- w

    // drop 1st element of each array
    return average(x[1 to n-1], weight[1 to n-1])
end-method
```

3. If an array contains n elements, what is the maximum number of recursive calls made by the binary search algorithm?

```
The first recursive call is AFTER the first split.
If n is even, array will have max n/2 elements after split
If n is odd, array will have (n – 1) / 2 elements after split

if n == 1: nor recursive call
if n == 2: (a, b), x <> a, call with b: one recursive call
if n == 3: (a, b, c), x <> b, call with a or c: one recursive call
if n == 4: split to (a) and (c, d): max of two recursive calls

Worst case is n = 2^k, max number of recursive calls is floor(log2(n)).
```

4. The expression m % n yields the remainder of m upon (integer) division by n. Define the greatest common divisor (GCD) of two integers x and y by:

gcd(x, y) = y	**if (y ≤ x and x%y == 0)**
gcd(x, y) = gcd(y, x)	**if (x < y)**
gcd(x, y) = gcd(y, x%y)	**otherwisef**

Write a recursive method to compute gcd(x,y).

```
method gcd(x, y)

    if x or y are not integers or less than 1 throw error

    if x < y then return gcd(y, x)

    // if we get here x > y
    define int m = x mod y
    if m == 0 then return y

    // m has to be less than y
    // y is less than x
    // so both args get smaller on recursion
    // guaranteeing that we hit the base case eventually
    return gcd(y, m)

end-method
```

5. A generalized Fibonacci function is like the standard Fibonacci function,, except that the starting points are passed in as parameters. Define the generalized Fibonacci sequence of f0 and f1 as the sequence gfib(f0, f1, 0), gfib(f0, f1, 1), gfib(f0, f1, 2), ..., where

gfib(f0, f1, 0) = f0
gfib(f0, f1, 1) = f1
gfib(f0, f1, n) = gfib(f0, f1, n-1) + gfib(f0, f1, n-2) if n> 1

Write a recursive method to compute gfib(f0,f1,n).

```
method gfib(f0, f1, n)

    if n < 0 or non-integer throw error
    if n == 1 return f1
    if n == 0 return f0

    return gfib(f0, f1, n-1) + gfib(f0, f1, n-2)

end-method
```

6. Ackerman's function is defined recursively on the nonnegative integers as follows:

a(m, n) = n + 1	if m = 0
a(m, n) = a(m-1, 1)	if m ≠ 0, n = 0
a(m, n) = a(m-1, a(m, n-1))	if m ≠ 0, n ≠ 0

Using the above definition, show that a(2,2) equals 7.

```
a(0, 1) = 2
```

```
a(1, 0) = a(0, 1) = 2

a(1, 1) = a(0, a(1, 0))
        = a(0, 2) = 3

a(1, 2) = a(0, a(1, 1))
        = a(0, 3) = 4

a(1, 3) = a(0, a(1, 2))
        = a(0, 4) = 5

a(1, 4) = a(0, a(1, 3))
        = a(0, 5) = 6

a(1, 5) = a(0, a(1, 4))
        = a(0, 6) = 7

a(2, 0) = a(1, 1) = 3

a(2, 1) = a(1, a(2, 0))
        = a(1, 3) = 5

a(2, 2) = a(1, a(2, 1))
        = a(1, 5) = 7
```

7. Show how to transform the following iterative procedure into a recursive procedure. *f(i)* is a method returning a logical value based on the value of *i*, and *g(i)* is a method that returns a value with the same attributes as *i*.

```
void iter(int n)
  {
     int i;
     i = n;
     while ( f(i) == TRUE ) {
        /* any group of statement that */
        /* does not change the value of i */
        i = g(i);
     } // end while
  } //end iter
```

```
Will need to assume that success calls to g cause i to converge to a value where f(i)
is FALSE.  The process doesn't return anything maybe it has side-effects like printing
to the console.

method iter_recursive(int i)
    // Base case
    if f(i) == false return

    // Any group of statements that do not change the value of i

    i = g(i)
    return iter_recursive(i)

end-method
```

8. Convert the following recursive program scheme into an iterative version that does not use a stack. *f(n)* is a method that returns TRUE or FALSE based on the value of n, and *g(n)* is a method that returns a value of the same type as *n* (without modifying *n*).

```
int rec(int n)
{
   if ( f(n) == FALSE ) {
      /* any group of statements that  do not change the value of n */
      return (rec(g(n)));
   }//end if
   return (0);
}//end rec
```

```
method rec_iterative(int n)
    loop while f(n) == FALSE
        // any group of statements that do not change the value of n
        n = g(n)
    end-loop
    return(0)
end-method
```