Assignment 3 – Recursion

Write pseudo-code for problems requiring code. Do not write Java, Python or C++. You are responsible for the appropriate level of detail. For the questions asking for justification, please provide a detailed mathematically oriented discussion. A proof is not required.

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 deals 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.

```
int sum(a,b)
        if b is greater than 0
                return a
        else return sum(a+1, b-1)
```

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.

```
main ()
        average (A)

average (array A)
        if A is valid
                averageHelper( A, 0)

averageHelper( arrary A, int index)
        if index + 1 < A.length //confirms not at end of the array
                if index is 0
                        return (A[0] + averageHelper(A, 1))/A.length
                return A[index] + averageHelper(A, index + 1)
        else if index +1 is equal to A.length //at the end of the array
                return A[index]
```

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

Max number of calls will be $\log_2 n + 1$

each time N will be cut in half and then one extra recursive call will be needed to determine the match

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)	**otherwise**

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

gcd(x,y)
if y is less than or equal to x and x % y == 0
return y
if x is less than y
return gcd (y, x)
return gcd(y, x%y)

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).

gfib(f0, f1, n)
if n is 0
return f0
if n is 1
return f1
return gfib(f0, f1, n-1) + gfib(f0, f1, n-2)

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(2,2) = 7
-> a(1, a(2,1)) = a(1,5) = 7
-> a(1,a(2,0)) = a(1,3) = 5
-> a(1,1) = 3
->a(0,a(1,0)) = a(0,2)
->a(0,1) = 2

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

a(1,5) = 7
-> a(0,a(1,4)) = a(0,6) = 7
->a(0,a(1,3)) = a(0,5) = 6

7. 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* itself).

```
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
```

```
while f(n) == FALSE //while f(n) is false, perform the following
      n = g(n) //set n to a new value based on g(n) for retest
return 0 //allowed to exit, not can retrun
```