CMPS 12B
Introduction to Data Structures
Fall 2016
Midterm Exam 1 Solutions

1. (20 Points) The Fibonacci function $F(n)$ is defined by $F(0)=0$, $F(1)=1$ and the recurrence formula $F(n)=F(n-1)+F(n-2)$ for $n \geq 2$.

   a. (10 Points) Write a *recursive* Java method with the following heading that calculates $F(n)$.

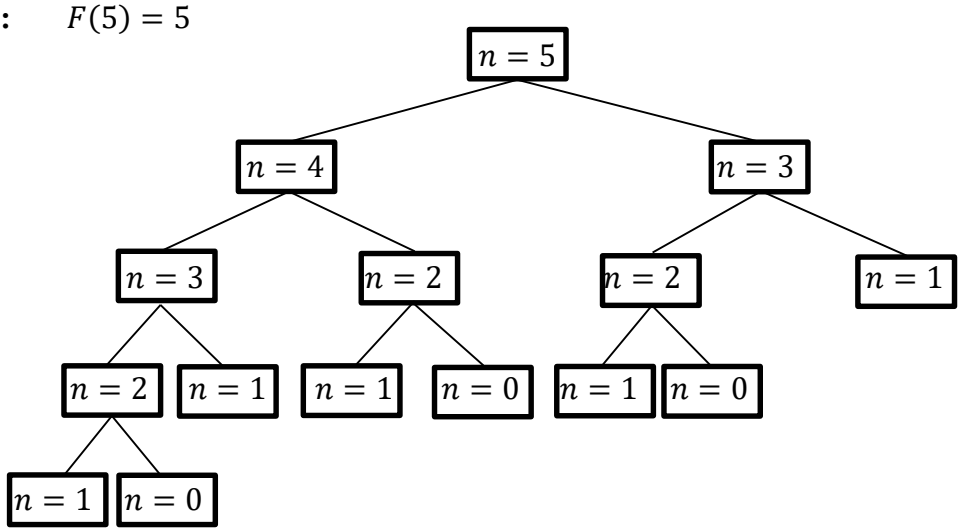
```
public static int F(int n){
   // your code starts here

   if( n==0 ){
      return 0;
   }else if( n==1 ){
      return 1;
   }else{
      return F(n-1)+F(n-2);
   }

}  // your code ends here
```

   b. (10 Points) Perform a box trace of the function call `F(5)`. Each box should state the value of *n* for the recursive call that it represents. What integer is returned?

Solution: $F(5)=5$



2. (20 Points) Write a recursive Java method that takes two non-negative integers *n* and *m* as input, then returns the sum of the integers from *n* to *m* (inclusive) if $n \leq m$, and returns 0 if $n > m$. Do this in two ways as described below.

   a. (10 Points) Determine the sum of integers from n to $m - 1$ recursively, then add *m* to the result. Call this function `sum1()` and fill in the code details below.

```
static int sum1(int n, int m){
   // your code starts here


   if( n>m ){
      return 0;
   }else{
      return sum1(n, m-1) + m;
   }


}  // your code ends here
```

   b. (10 Points) Split the sequence of integers from *n* to *m* (roughly) in half, recur on the two half-sequences, then add the results. Call this function `sum2()` and fill in the code details below. Hint: model this function on `mergeSort()`.

```
static int sum2(int n, int m){
   // your code starts here


   if( n>m ){
      return 0;
   }else if( n==m ){
      return n;
   }else{
      int k = (n+m)/2;
      return sum2(n, k) + sum2(k+1, m);
   }


}  // your code ends here
```

3. (20 Points) Recall the IntegerList ADT whose interface is reproduced below. (Comments are included to remind you what each method does.)

```
public interface IntegerListInterface{
   public boolean isEmpty();  // return true iff list has no elements
   public int size();         // return number of elements in list
   public int get(int index); // return list element at index
   public void add(int index, int newItem); // insert newItem at index
   public void remove(int index);  // delete element at index
   public void removeAll();   // reset list to empty state
}
```

a. (10 Points) Write a `static void` method called `swap()` with the following heading that interchanges the items currently in positions `i` and `j` of the List. Use only the ADT operations above to do this.

```
static void swap(IntegerList L, int i, int j){
   // your code starts here

   int a = L.get(i);
   int b = L.get(j);
   L.add(i,b);
   L.remove(i+1);
   L.add(j,a);
   L.remove(j+1);

}  // your code ends here
```

b. (10 Points) Write a `static void` method called `reverse()` with the following heading that reverses the order of the items in the List. Use only the ADT operations above and function `swap()` from part (a) to do this.

```
public static void reverse(IntegerList L){
   // your code starts here

   if( L.size()>1 ){
      int i=1;
      int j=L.size();
      while(i<j){
         swap(L,i,j);
         i++;
         j--;
      }
   }

}  // your code ends here
```

4. (20 Points) Determine the output of the following java program. Print the program output on the lines below exactly as it would appear on the screen. Note more lines are included than are actually needed. (Hint: it will help to perform a detailed box trace of recursive calls to function `foo()`, keeping track of all local variables.)

```
// Problem4.java
class Problem4 {
   static double foo(double x, int n){
      double y = x+n;
      System.out.println("in foo: "+ x +", "+ n);
      x = x/2;
      if( x > n ){
         y = foo(x, n-1);
      }
      System.out.println("y = "+ y);
      return x;
   }
   public static void main(String[] args){
      System.out.println(foo(100.0, 10));
   }
}
```

Program Output:

```
in foo: 100.0, 10
in foo: 50.0, 9
in foo: 25.0, 8
in foo: 12.5, 7
y = 19.5
y = 6.25
y = 12.5
y = 25.0
50.0
```

5. (20 Points) Given the Java classes Node and LinkedList below, complete the definitions of the *recursive* functions `printForward()` and `printBackward()` so as to print out the elements in a list from head to tail, and from tail to head respectively. Each function should print a space separated list of integers to the standard output stream.

```
// file: Node.java
class Node{
   int item;
   Node next;
   Node(int x){
      item = x;
      next = null;
   }
}
// file: LinkedList.java
class LinkedList{
   private Node head;
   LinkedList(){ // creates an empty LinkedList
      head = null;
   }

   void printForward(Node H){
      // your code starts here

      if( H!=null ){
         System.out.print(H.item+" ");
         printForward(H.next);
      }

      // your code ends here
   }

   void printBackward(Node H){
      // your code starts here

      if ( H!=null ){
         printBackward(H.next);
         System.out.print(H.item+" ");
      }

      // your code ends here
   }

}
```