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 and returns $F(n)$.

   ```java
   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:** $F(5) = 5$

   ![Fibonacci Tree Diagram]

   The diagram shows the recursive calls made by the method $F(5)$. Each node represents a call to the method with a specific value of $n$, and the branches illustrate the recursive nature of the Fibonacci function.
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 \( \text{sum1()} \) and fill in the code details below.

```java
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 \( \text{sum2()} \) and fill in the code details below. Hint: model this function on \( \text{mergeSort()} \).

```java
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.)

```java
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.

```java
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.

```java
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) Given classes Node and NodeTest defined below, answer the following questions.
   a. (6 Points) Trace function main() in NodeTest.java up to point (a) and draw a picture of the
      resulting linked list data structure.
   b. (6 Points) Trace execution of main() up to point (b) and write the output as it would appear on the
      screen. Note this is a single line of output, so it will fit in the space provided.
   c. (8 Points) Write instructions that will insert a new Node with item value 5 into the front of the list
      (before the 4).

// file: Node.java
class Node{
    int item;
    Node next;
    Node(int x){
        item = x;
        next = null;
    }
}
// file: NodeTest.java
class NodeTest{
    public static void main(String[] args){
        Node head = new Node(4);
        Node N = head;
        N.next = new Node(3);
        N = N.next;
        N.next = new Node(2);
        N = N.next;
        N.next = new Node(1);
        // part (a) refers to this point in the code

        head 4 3 2 1

        N = head;
        while(N != null){
            System.out.print(N.item + " ");
            N = N.next;
        }
        System.out.println();
        // part (b) refers to this point in the code

        // output: 4 3 2 1

        // part (c) refers to this point in the code
        // your code begins here

        N = new Node(5);
        N.next = head;
        head = N;

        // your code ends here
    }
}
5. (20 Points) Given the Node class in problem 4 above, and a linked list based on that class, write functions called `insertFront()` and `insertBack()` described below.

a. (10 Points) Function `insertFront()` will place a new Node with item \( x \) at the front of a list headed by the global Node reference `head`.

```java
static void insertFront(int x){
    // your code starts here

    Node N = new Node(x);
    N.next = head;
    head = N;

    // your code ends here
}
```

b. (10 Points) Function `insertBack()` will place a new Node with item \( x \) at the back of a list headed by the global Node reference `head`.

```java
static void insertBack(int x){
    // your code starts here

    if(head==null){   // list is empty
        head = new Node(x);
    }else{            // list is not empty
        Node N = head;
        while( N.next!=null ){
            N = N.next;
        }
        N.next = new Node(data);
    }

    // your code ends here
}
```