1. Recall the recursive function $C(n, k)$ in the class `BinomialCoefficients` discussed in lecture and posted on the webpage. Write a box trace of the function call $C(5, 3)$. Use this trace to find the value of $C(5, 3)$. Notice that in the full recursion tree for $C(5, 3)$, the value $C(3, 2)$ is evaluated 2 times, and $C(2, 1)$ is evaluated 3 times. Suggest a modification to the function that would allow it to avoid computing the same values multiple times. Carry out your suggestion by writing the Java code.

**Solution to second question:**
**Suggested modification:** when $C(n, k)$ is computed for the first time (for a particular $n$ and $k$), save the value in a static 2-dimensional array for later re-use. If the value $C(n, k)$ is needed at some later time, look it up in the array instead of computing it again.

```java
class Problem1{
    static int C(int n, int k, int[][] BinCoef){
        if( BinCoef[n][k]!=0 ){
            return BinCoef[n][k];
        } else if( k==0 || k==n ){
            return 1;
        } else{
            BinCoef[n][k] = C(n-1,k-1, BinCoef)+C(n-1,k, BinCoef);
            return BinCoef[n][k];
        }
    }
    public static void main(String[] args){
        int n = 10, k = 6, i, j;
        int[][] BC = new int[n+1][k+1];
        for(i=0; i<=n; i++){
            for(j=0; j<=k; j++){
                BC[i][j] = 0;
            }
        }
        System.out.println(C(n, k, BC));
    }
}
```

2. Write a recursive function called $\text{sum}(n)$ that computes the sum of the integers from 1 to $n$. Hint: emulate the factorial function discussed in lecture (replace multiplication by addition). Modify your answer to recursively compute the sum of the integers from $n$ to $m$, where $n \leq m$ (return 0 if $n > m$).

**Solution to second question:**
```java
static int sum(int n, int m){
    if( n<=m ){
        return sum(n, m-1) + m;
    } else{
        return 0;
    }
}
```
3. Write recursive functions that determine the sum of the elements in an int array. Do this in 3 ways.
   a. Write a recursive function `sumArray1(int[] A, int n)` that returns the sum of the leftmost $n$ elements of $A[]$. Get the $n^{th}$ element from the left, compute the sum of the leftmost $(n - 1)$ elements recursively, then return the sum.
   b. Write a recursive function `sumArray2(int[] A, int n)` that returns the sum of the rightmost $n$ elements of $A[]$. Get the $n^{th}$ element from the right, compute the sum of the rightmost $(n - 1)$ elements recursively, then return the sum.
   c. Write a recursive function `sumArray3(int[] A, int p, int r)` that returns the sum of the subarray $A[p...r]$. Hint: use `MergeSort()` as a model.

Solution:
```java
class Problem3{
  // return the sum of the leftmost n elements in A
  static int sumArray1(int[] A, int n){
    int a, b;
    if( n<=0 ){                // if array is empty
      return 0;                  // return zero
    }else{                     // else if array is non-empty
      a = A[n-1];                // get nth element from the left
      b = sumArray1(A, n-1);     // compute sum of leftmost (n-1) elements
      return a+b;                // return the sum
    }
  }
  // return the sum of the rightmost n elements in A
  static int sumArray2(int[] A, int n){
    int a, b;
    if( n<=0 ){                // if array is empty
      return 0;                  // return zero
    }else{                     // else if array is non-empty
      a = A[A.length-n];         // get nth element from the right
      b = sumArray2(A, n-1);      // compute sum of rightmost (n-1) elements
      return a+b;                // return the sum
    }
  }
  // return sum of the subarray A[p...r]
  static int sumArray3(int[] A, int p, int r){
    int a, b, q;
    if( p<r ){
      q = (p+r)/2;
      a = sumArray3(A, p, q);
      b = sumArray3(A, q+1, r);
      return a+b;
    }else{
      return A[p];
    }
  }
  public static void main(String[] args){
    int[] X = {1, 2, 3, 4, 5, 6, 7};
    System.out.println(sumArray1(X, X.length));
    System.out.println(sumArray2(X, X.length));
    System.out.println(sumArray3(X, 0, X.length-1));
  }
}
```
4. Write a modification of the recursive function `BinarySearch()` that prints out the sequence of array elements that are compared to the target.

**Solution:**

```java
static int BinarySearch(int[] X, int p, int r, int target){
    if( p<=r ){
        int q = (p+r)/2;
        System.out.print(X[q]+" ");
        if( target==X[q] ){
            return q;
        }else if( target<X[q] ){
            return BinarySearch(X, p, q-1, target);
        }else{  // target>X[q]
            return BinarySearch(X, q+1, r, target);
        }
    }else{
        return -1;
    }
}
```

6. Perform a box trace of the following recursive function for the input $n=100$, $level=0$, and determine the output. What does the function do?

```java
static void doSomething(int n, int level){
    if(n>0){
        if(n/8>0){
            doSomething(n/8, level+1);
        }else{
            System.out.print(n%8);
        }
    }else if( level==0 ){
        System.out.print(0);
    }
}
```

**Solution:**

This function prints the octal digits of the input $n$. The purpose of the function parameter `level` is to keep track of the recursion depth, and print the digit zero only in the case when $n = 0$ at the very top level. This is necessary since the value $n = 0$ is used to halt the recursion at deeper levels. Note that this is a partial solution to the exercises mentioned in the notes from 4-4-19 pages 8-12. The box trace follows.

```
Print 4
n = 100, level = 0
  |
  v
Print 4
n = 12, level = 1
  |
  v
Print 1
n = 1, level = 2
```

7. Use what you learned in problem 6 above to create a recursive function called `integerToString()` that returns a String representation of the integer \( n \) expressed in base \( b \). For instance, the function call `integerToString(100, 8, 0)` would return the String “144”, which was printed in problem 6. Your function need not allow bases greater than 10 (although that would be a good exercise), but it should deal correctly with the input \( n = 0 \) (i.e. return the digit "0" in that case).

**Solution:**
The following full Java program defines `integerToString()` and tests it on various bases. It uses a helper function called `digit()` that produces digits in bases greater than 10.

```java
class Problem7{
    // integerToString()
    // Returns a string representation of n in base b, where b is in the range
    // 2 <= b <= 36. The int variable level keeps track of the recursion depth.
    static String integerToString(int n, int b, int level){
        String s="";
        if( n>0 ){
            if( n>=b ){
                s = integerToString(n/b, b, level+1);
            }
            return s + digit(n%b);
        }else if( level>0 ){
            return s;
        }else{
            return String.valueOf(0);
        }
    }

    // digit()
    // pre: 0 <= d <= 35
    // Returns a printable ascii char from the set {0, 1,..., 9, A,..., Z}
    // representing d in the range 0 <= d <= 35.
    static String digit(int d){
        if( d<0 || d>35 ){
            System.err.println("no digit available to represent "+d);
            System.exit(1);
        }
        if( d<10 ){
            return String.valueOf(d);
        }else{
            return String.valueOf((char)(d+55));  // 65 <= (d+55) <= 90
        }
    }

    public static void main(String[] args){
        int n = 43981;
        //int n = 0;
        //int n = 100;
        for(int b=2; b<=36; b++){
            System.out.println("base = "+b+"\t"+integerToString(n, b, 0));
        }
    }
}
```
8. Recall the IntegerList ADT discussed in class whose states were the finite integer sequences, and whose operations were isEmpty(), size(), get(), add(), remove(), and removeAll(). Write the methods described below using only these six ADT operations. In other words you are writing methods belonging to a client of IntegerList.

   a. Write a static void method called swap(IntegerList L, int i, int j) that will interchange the items currently at positions i and j of the List.

   b. Write a static int method called search(IntegerList L, int x) that will perform a linear search of L for the target x. search() will return the List index where x was found, or it will return 0 if no such index exists. (Recall List indices range from 1 to size().)

   c. Write a static void method called reverse(IntegerList L) that reverses the order of the items in L.

Solution to part a:
static void swap(IntegerList L, int i, int j){
    int a = L.get(i);
    int b = L.get(j);
    L.remove(i);
    L.add(i, b);
    L.remove(j);
    L.add(j, a);
}

Solution to c:
static reverse(IntegerList L){
    int i=1, j=L.size();
    while( i<j ){
        swap(L, i, j);
        i++;
        j--;
    }
}