For this Assignment, none of your methods may call auxiliary methods (except itself) unless otherwise stated.

Problem 1 Write a divide and conquer (recursive, no loops) algorithm which takes positive integers *n* and *k*, as well as a one-dimensional array *D* (indexed from 1 to *k*) whose values are sorted in ascending order. The return value is the number of ways to make change for *n* cents using coins of denomination $D[1], D[2], D[3], \dots, D[k]$. **You may NOT assume that $D[1] = 1$**.

```
int numWaysChange(int n, int k, int [] D, int g)  {. . .}
```

Problem 2 Write the binary search algorithm using divide and conquer (recursive, no loops), which takes an array *A*, indices *low* and *high*, as well as a value *x*. If *x* is found in *A* between indices *low* and *high*, return the index where *x* is located. Otherwise return -1. Your algorithm must ***run in time $O(\lg n)$***, where *n* = the number of elements in the array between indices *low* and *high*.

```
int binarySearch(int [] A, int low, int high, int x)  {. . .}
```

Problem 3 Write the method `partition`, which takes an array *A* and indices *low* and *high*, and ***partitions*** the array using *A*[*low*] as the pivot value. In other words, this method changes *A* by putting the value *A*[*low*] at some index *p* and arranging the values in *A* so that all values in *A* which are greater than the value *A*[low] are located between indices *p* + 1 and *high*, and all values in *A* less than (or equal to) *A*[low] are located between indices *low* and *p* – 1. The method returns the index *p*. Your algorithm must ***run in time $O(n)$***.

```
int partition(int [] A, int low, int high)  {. . .}
```

Problem 4 Write the method below which takes an array *A* and indices *low* and *high*. The method returns the index of the minimum element in array *A* between indices *low* and *high*. You must use divide and conquer (recursive, no looping statements). What is the running time of your algorithm?

```
int indexOfMinElt(int [] A, int low, int high)  {. . .}
```

Problem 5 Write a divide and conquer algorithm (recursive, no looping statemnts) which sorts an array between indices low and high. You may call the method `indexOfMinElt` from the previous problem, but no other methods (other than itself). What is the running time of your algorithm?

```
void sortRec(int [] A, int low, int high)  {. . .}
```

Problem 6 You are given 9 indentical looking coins: $c_1, c_2, c_3, \dots, c_9$. One of them is a counterfeit coin and weighs a tiny bit more than the rest, but nothing you detect on your own. You have a balance scale. You put coins into the left and right basket of the balance scale. Then you put in $1, which allows the scale to now tip, the heavier side going down, the lighter side going up. If both sides are the same weight, the scale remains level. How can you find the counterfeit coin using at most $2 (i.e., at most 2 weighings)? Describe this in text for submission.

Problem 7 Consider the scenario of the problem above, but with n coins instead of 9 (assume n is a power of 3). Describe (in words) an algorithm for finding the couterfeit coin using ***at most*** $\log_3 n$ dollars (weightings). (See the last problem for extra credit for implementing this algorithm in Java.)

Problem 8 Write a divide and conquer algorithm (recursive, no looping statements) that returns the average of the real numbers in array *A* that are located between indices *low* and *high* inclusive. Assume that number of array elements located between indices *low* and *high* inclusive is a power of 2, and that *average* is never called with *low* > *high*. Your algorithm may NOT alter array *A*.

```
double average(double [] A, int low, int high)  {. . .}
```

Problem 9 For this problem, use pseudocode only—do not progam it in Java. Write the divide and conquer algorithm `prod` which takes two `BigInteger`'s and returns the `BigInteger` product of the two. Your algorithm must ***run in time*** $O(n^{1.585})$. Note: $\log_2 3 \approx 1.58496$. You MAY call the method `BigInteger.multiply` ***only*** in the base case ***or*** with arguments `Ten_to_m` and `Ten_to_2m` (these are defined for you). Otherwise, you may **NOT call** `BigInteger.multiply` You MAY call the following `BigInteger` methods:

- `BigInteger.add`
- `BigInteger.subtract`
- `BigInteger.pow`
- `BigInteger.divide`
- `BigInteger.mod`

```
BigInteger prod(BigInteger u, BigInteger v)  {. . .}
```

Problem 10 Trace through the largeInt multiplication algorithm to find the product of $u = 2132$ and $v = 1234$. Show each call that is made. What is the product? How many single-digit multiplicatons were done while computing this product? Follow this format:

<table>
<tr>
<td>
call # 1:
prod(2132, 1345)
n = 4
m = 2
x = 21
y = 32
w = 13
z = 45
r = prod(x + y, w + z)
= prod(53, 58) = _____ (See call #2)
p = prod(x, w)
= prod(21, 13) = _____ (See call #3)
q = prod(y, z)
= prod(32, 45) = _____ (See call #4)
return p * 10^4 + (p – r – q) * 10^2 + q = _______
</td>
<td>
call # 2:
prod(53, 58)
n = 2
m = 1
…
…
return p * 10^4 + (p – r – q) * 10^2 + q = _______
(put the value in the this blank in the blank for call # 2)
</td>
</tr>
</table>

Problem 11 Write a divide and conquer algorithm (recursive, no loops) to compute the number of *k*-element subsets of the set {1, 2, 3, …, *n*}, i.e., the number of ways to choose *k* items from *n* items.

```
int combinations(int n, int k)  {. . .}
```

Problem 12 For this problem a *Point* is an object with two public fields—an *x*-coordinate and a *y*-coordinate. Write a ***brute force*** algorithm (try all possibilities) to find the closest pair of points:

```
int [] closestPair(int n, Point [] Q)
```

The array *Q* is indexed from 1 to *n*, where $Q[i]$ is the *i*th point, and $Q[i].x$ is its *x*-coordinate and $Q[i].y$ is its *y*-coordinate. The *Euclidean distance* between two points (x_1, y_1) and (x_2, y_2) is given by:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Return array `ans` of length 2 with values *i* and *j* such that the closest two points in Q are $Q[i]$ and $Q[j]$.

Problem 13 (+5 pts extra credit) Using Java, write (and test!) a divide and conquer algorithm which finds the counterfeit coin from among n coins. You may assume n is a power of 3. More specifically, write the method

```
int counterfeit(int n)  {. . .}
```

which returns the index i such that coin c_i is the counterfeit coin from the set of coins $\{c_{1,}\ c_{2,}\ c_{3,} \ldots, c_n\}$. You algorithm may make ***at most*** $\log_3 n$ calls to the following (provided) method:

```
int balance(int low1, int high1, int low2, int high2)
```

This method "weighs" the group of coins: $c_{\text{low1}}, c_{\text{low1+1}}, c_{\text{low1+2}}, \ldots, c_{\text{high1}}$ (call this group G_1) against the group of coins: $c_{\text{low2}}, c_{\text{low2+2}}, c_{\text{low2+2}}, \ldots, c_{\text{high2}}$ (call this group G_2). The return value of `balance` is as follows:

1 if G_1 is heavier than G_2,
2 if G_2 is heavier than G_1, and
0 if G_1 and G_2 weigh the same.

You must submit your Java program and the results of testing it to get the extra credit. You may use Math.pow and Math.sqrt for this program.
