CS 142 Sample Final Exam #1

1. Big-Oh Analysis

Give a tight bound of the runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable *N*.

a)
```
int sum = 0;
for (int i = 1; i <= N + 3; i++) {
    for (int j = 1; j <= N * N; j++) {
        sum++;
    }

    sum++;
}
System.out.println(sum);
```

b)
```
int sum = 0;
for (int i = 1; i <= N - 3; i++) {
    for (int j = 1; j <= i + 4; j += 2) {
        sum++;
    }

    sum++;
}

for (int i = 1; i <= 100; i++) {
    sum++;
}
System.out.println(sum);
```

c)
```
int sum = 0;
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= 1000; j += 2) {
        sum++;
    }
}

for (int k = -50; k <= -1; k++) {
    sum++;
}
System.out.println(sum);
```

d)
```
List<Integer> set = new ArrayList<>();
for (int i = 1; i <= N * 2; i *= 2) {
    set.add(i);
}

for (int k : set) {
    System.out.println(k);
}
System.out.println("done!");
```

e)
```
List<Integer> list = new ArrayList< >();
for (int i = 1; i <= N; i++) {
    list.add(i);
}

while (!list.isEmpty()) {
    list.remove(0);
}
System.out.println("done!");
```

f)
```
List<Integer> list = new ArrayList<>();
for (int i = 1; i <= N / 2; i++) {
    list.add(i);
}
for (int i = 1; i <= N / 2; i++) {
    list.add(0, i);
}

while (!list.isEmpty()) {
    list.remove(0);
    list.remove(list.size() - 1);
}
System.out.println("done!");
```

2. Array of Arrays Mystery

Write the output produced by the program shown below.

```
public class Example {
      public static void main(String[] args) {
          int[][] v = {{ 1, 3,  54, 23, 2},
                       {34, 4,  22, 2,  77},
                       {0,  6,  35, 61, 2},
                       {5,  7,  8,  9,  123},
                       {32, 14, 44, 17, 23 }};

          mystery(v, true);
      }

      public static void mystery(int[][] v, boolean dir) {
          Int[][] result = new int[v.length] [v[0].length];
          for (int row = 0; row < v.length; row++) {
              for (int col = 0; col < v[0].length; col++) {
                  int px = v[ row ][ col ];
                  if (dir) {
                      result[ v[ 0 ].size() - 1 - col ][ row ] = px;
                  } else {
                      result[ col ][ v.size() - 1 - row ] = px;
                  }
                  dir = !dir;
              }
          }
          System.out.println(Arrays.toString(result));
      }
}
```

3. **Recursive Tracing**. For each call to the following method, indicate what value is returned:

```
public static int mystery(int n) {
    if (n < 0) {
        return -mystery(-n);
    } else if (n < 10) {
        return (n + 1) % 10;
    } else {
        return 10 * mystery(n / 10) + (n + 1) % 10;
    }
}
```

Call	Value Returned
`mystery(7)`	
`mystery(42)`	
`mystery(385)`	
`mystery(-790)`	
`mystery(89294)`	

4. **Recursive Programming**. Write a recursive method `digitMatch` that accepts two non-negative integers as parameters and that returns the number of digits that match between them. Two digits match if they are equal and have the same position relative to the end of the number (i.e., starting with the ones digit). In other words, the method should compare the last digits of each number, the second-to-last digits of each number, the third-to-last digits of each number, and so forth, counting how many pairs match. For example, for the call of `digitMatch(1072503891, 62530841)`, the method would compare as follows:

```
1 0 7 2 5 0 3 8 9 1
    | | | | | | | |
    6 2 5 3 0 8 4 1
```

The method should return 4 in this case because 4 of these pairs match (2-2, 5-5, 8-8, and 1-1). Below are more examples:

Call	Value Returned
`digitMatch(38, 34)`	1
`digitMatch(5, 5552)`	0
`digitMatch(892, 892)`	3
`digitMatch(298892, 7892)`	3
`digitMatch(380, 0)`	1
`digitMatch(123456, 654321)`	0
`digitMatch(1234567, 67)`	2

Your method should throw an `IllegalArgumentException` if either of the two parameters is negative. You are not allowed to construct any structured objects other than `String`s (no array, `List`, `Scanner`, etc.) and you may not use any loops to solve this problem; you must use recursion.

5. **Inheritance and Comparable Programming**. You have been asked to extend a pre-existing class `Point` that represents 2-D (x, y) coordinates. The `Point` class includes the following constructors and methods:

Constructor/Method	Description
`public Point()`	constructs the point (0, 0)
`public Point(int x, int y)`	constructs a point with the given x/y coordinates
`public void setLocation(int x, int y)`	sets the coordinates to the given values
`public int getX()`	returns the x-coordinate
`public int getY()`	returns the y-coordinate
`public String toString()`	returns a `String` in standard "(x, y)" notation
`public double distanceFromOrigin()`	returns the distance from the origin (0, 0), computed as the square root of $(x^2 + y^2)$

You are to **define a new class called `Point3D` that extends this class through inheritance**. It should behave like a `Point` except that it should be a 3-dimensional point that keeps track of a z-coordinate. You should provide the same methods as the superclass, as well as the following new behavior.

Constructor/Method	Description
`public Point3D()`	constructs the point (0, 0, 0)
`public Point3D(int x, int y, int z)`	constructs a point with given x/y/z coordinates
`public void setLocation(int x, int y, int z)`	sets coordinates to the given values
`public int getZ()`	returns the z-coordinate

Some of the existing behaviors from `Point` should behave differently on `Point3D` objects:

- When the original 2-parameter version of the `setLocation` is called, the 3-D point's x/y coordinates should be set as specified, and the z-coordinate should be set to 0.
- When a 3-D point is printed with `toString`, it should be returned in an "(x, y, z)" format that shows all three coordinates.
- A 3-D point's distance from the origin is computed using all three coordinates; it is equal to the square root of $(x^2 + y^2 + z^2)$.

You must also **make `Point3D` objects comparable to each other using the `Comparable` interface**. 3-D points are compared by x-coordinate, then by y-coordinate, then by z-coordinate. In other words, a `Point3D` object with a smaller x-coordinate is considered to be "less than" one with a larger x-coordinate. If two `Point3D` objects have the same x-coordinate, the one with the lower y-coordinate is considered "less." If they have the same x and y-coordinates, the one with the lower z-coordinate is considered "less." If the two points have the same x, y, and z-coordinates, they are considered to be "equal."

6. **Searching and Sorting**.

(a) Suppose we are performing a **binary search** on a sorted array called `numbers` initialized as follows:

```
// index          0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
int[] numbers = {-9, -6, -2, -1,  0,  1,  3,  4,  5,  7,  9, 10, 12, 19, 23, 26};

// search for the value 1
int index = binarySearch(numbers, 1);
```

Write the indexes of the elements that would be examined by the binary search (the `mid` values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

- Indexes examined: _____
- Value Returned: _____

(b) Write the state of the elements of the array below after each of the first 3 passes of the outermost loop of the selection sort algorithm.

```
int[] numbers = {37, 29, 19, 48, 23, 55, 74, 12};
selectionSort(numbers);
```

(c) Trace the complete execution of the merge sort algorithm when called on the array below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-arrays that are created by the algorithm and show the merging of sub-arrays into larger sorted arrays.

```
int[] numbers = {37, 29, 19, 48, 23, 55, 74, 12};
mergeSort(numbers);
```

7. 2D Arrays

Write a method called `findMax` that takes a two-dimensional array as a parameter and returns the number of the row that sums to the greatest value. For example, if you had the following array of arrays:

```
int[][] list = {{1, 2, 3}, {2, 3, 3}, {1, 3, 3}}
```

The first row would be 6, the second 8 and the third 7. The method would therefore return 1.

You can assume the passed in array of arrays has at least one row and one column. You cannot assume that it is square.

8. Programming

Expect to see one more programming question here on the actual exam. It will involve lists and/or strings. It could involve something else already on this test like recursion, inheritance, sorting, etc. However, it will be similar to problems you have previously been assigned on Code Step By Step.