CS 142 Sample Final Exam #2

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 = 0; i < N * 2; i++) {
    for (int j = 0; j < i / 3; j++) {
        for (int k = 0; k < j*j; k+=3) {
            sum++;
        }
    }
}
System.out.println(sum);
```

b)
```
int sum = 0;
for (int i = 1; i < N; i *= 2) {
    for (int j = 1; j < N; j *= 2) {
        sum++;
    }
}
System.out.println(sum);
```

c)
```
List<Integer> set1 = new ArrayList<>();
for (int i = 0; i < N; i++) {
    set1.add(i);
}
for (int i = 0; i < N; i++) {
    set1.set(i, i / 2);
}
System.out.println("done!");
```

d)
```
List<Integer> list = new ArrayList<>();
for (int i = 0; i < N; i++) {
    list.add(i);
}
List<Integer> set = new ArrayList<>();
for(int i = 0; i < list.size(); i++) {
    if(!set.contains(list.get(i))) {
    set.add(list.get(i));
}
System.out.println("done!");
```

e)
```
List<Integer> list1 = new ArrayList<>();
for (int i = 0; i < N; i += 2) {
    list1.add(i);
}
List<Integer> list2 = new ArrayList<>();
for (int i = 0; i < N; i++) {
    list2.add(0, list1.remove(0));
}
System.out.println("done!");
```

f)
```
int sum = 0;
for (int i = 0; i < N * 2; i++) {
    for (int j = 0; j < 10000; j++) {
        for (int k = 0; k < j*j; k++) {
            sum++;
        }
    }
}
System.out.println(sum);
```

2. Array of Arrays Mystery

Write the output produced by the program shown below.

```
public class Example2 {
      public static void main(String[] args) {
          int[][] numbers = {{0, 1, -23, 99, 13, 1},
                             {2, 1, 1, 1, 2, 1},
                             {0, 51, 0, 5, 0, 5},
                             {3, 49, 13, 4, 13, 52},
                             {1, 49, 49, 4, 14, 4}};

          System.out.println(Arrays.toString(mystery(v)));
      }

      public static void mystery(int[][] vec) {
          for (int r = 1; r < vec.length - 1; r++) {
              for (int c = 1; c < vec[ r ].length - 1; c++) {
                  if (vec[r - 1][c - 1] % 10 < 5) {
                      vec[r - 1][c]--;
                  } else {
                      vec[c][r - 1] += vec[r - 1][c];
                  }
              }
          }
      }
}
```

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

```
public static void mystery(int x, int y) {
    if (x > y) {
        System.out.print("*");
    } else if (x == y) {
        System.out.print("=" + y + "=");
    } else {
        System.out.print(y + " ");
        mystery(x + 1, y - 1);
        System.out.print(" " + x);
    }
}
```

Call	Output
`mystery(3, 3);`	
`mystery(5, 1);`	
`mystery(1, 5);`	
`mystery(2, 7);`	
`mystery(1, 8);`	

4. **Recursive Programming**. Write a recursive method `repeat` that accepts a string *s* and an integer *n* as parameters and that returns a string consisting of *n* copies of *s*. For example:

Call	Value Returned
`repeat("hello", 3)`	`"hellohellohello"`
`repeat("this is fun", 1)`	`"this is fun"`
`repeat("wow", 0)`	`""`
`repeat("hi ho! ", 5)`	`"hi ho! hi ho! hi ho! hi ho! hi ho! "`

You should solve this problem by concatenating strings using the `+` operator. String concatenation is an expensive operation, so it is best to minimize the number of concatenation operations you perform. For example, for the call `repeat("foo", 500)`, it would be inefficient to perform 500 different concatenation operations to obtain the result. Most of the credit will be awarded on the correctness of your solution independent of efficiency. The remaining credit will be awarded based on your ability to minimize the number of concatenation operations performed.

Your method should throw an `IllegalArgumentException` if passed a negative value for *n*. 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 `Date` that represents calendar dates such as March 19. The `Date` class includes these constructors and methods:

Constructor/Method	Description
`public Date(int month, int day)`	constructs a `Date` object with the given month/day
`public int getMonth()`	returns the month
`public int getDay()`	returns the day
`public void setMonth(int month)`	sets month to a new value
`public void setDay(int day)`	sets day to a new value
`public int daysInMonth(int month)`	returns number of days in the given month (examples: 4 → 30; 10 → 31; 2 → 28)
`public void nextDay()`	advances to next date, wrapping month if needed (3/19 → 3/20; 1/31 → 2/1; 12/31 → 1/1)
`public String toString()`	returns string version of date, such as `"03/19"`

You are to **define a new class called `CalendarDate` that extends this class through inheritance**. It should behave like a `Date` except that it should also keep track of the year. You should provide the same methods as the superclass, as well as the following new behavior:

Constructor/Method	Description
`public CalendarDate(int year, int month, int day)`	constructs a `CalendarDate` object with the given year/month/day
`public int getYear()`	returns the year
`public void setYear(int year)`	sets year to a new value

Some of the existing behaviors from `Date` should behave differently on `CalendarDate` objects:

- When a `CalendarDate` is printed with `toString`, it should be returned in a year/month/day format such as `"2009/03/19"`. Note that the year comes first. There should be a leading 0 if necessary if the month and/or day is less a single-digit number.
- When advancing a `CalendarDate` object to the next date using `nextDay`, if this is the last date of the year (December 31st), you should wrap the object to the next year. For example, the next day after December 31st, 2009 is January 1st, 2010.

You must also **make `CalendarDate` objects comparable to each other using the `Comparable` interface**. Calendar dates are compared by year, then by month, then by day. In other words, a `CalendarDate` object with a smaller year is considered to be "less than" one with a larger year. If two objects have the same year, the one with the lower month is considered "less." If they have the same year and month, the one with the lower day is considered "less." If the two objects have the same year, month, and day, they are considered to be "equal."

You may assume that all year/month/day values passed to your methods and constructors are valid. You should not worry about leap years for this problem; assume that February always has 28 days.

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
int[] numbers = {-1,  0,  5, 11, 18, 29, 53, 57, 61, 64, 78, 82, 85, 89, 93};

// search for the value 86
int index = binarySearch(numbers, 86);
```

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 = {8, 5, -9, 14, 0, -1, -7, 3};
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 = {8, 5, -9, 14, 0, -1, -7, 3};
mergeSort(numbers);
```

7. Array of Arrays Programming

Write a method called `numUnique` that takes an array of arrays of integers as a parameter and returns the number of unique values stored in it. For example, if you have the following array of arrays:

```
int[][] lis = {{1, 2, 3}, {4, 3, 2, 1}, {6, 7, 7}, {8}};
```

a call to `numUnique(lis)` should return 7. You may create one other data structure to help you solve this problem.

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.