CSE 373, Spring 2012
Homework #2: Algorithm Analysis (40 points)
Due Wednesday, April 11, 2012, Beginning of Class

Turn in your answers to #1 - #4 on paper. Please write neatly and clearly. Staple multiple pages together and write your name on the top of every page.

Turn in your answer to #5 electronically using the link off of the homework webpage.

1. (4 points) Show that the function $\log_2(9N)$ is $O(\log_2 N)$. You will need to use the definition of $O(f(n))$ to do this. In other words, find values for c and n_0 such that the definition of Big-Oh holds true as we did with the examples in lecture. For full credit, show the steps you took to arrive at your c and n_0.

2. (6 points) (adapted from Weiss 2.1) Order the following functions by growth rate:

N^2, $N \log N$, $2/N$, 2^N, 92, $N^2 \log N$, $N!$, $N^{1.5}$, N^3, $\log N$, $N \log (N^2)$, $4^{\log N}$, N

Indicate which functions grow at the same rate. Recall that a function $f(N)$ grows at the same rate as function $g(N)$ if $f(N) = \Theta(g(N))$.

3. (8 points) (from Weiss 2.2) You do not need to prove an item is true (just saying true is enough for full credit), but you must give a counter example in order to demonstrate an item is false if you want full credit. To give a counter example, give values for $T_1(N)$, $T_2(N)$, and $f(N)$ for which the statement is false.

Suppose $T_1(N) = O(f(N))$ and $T_2(N) = O(f(N))$. Which of the following are true?
a. $T_1(N) + T_2(N) = O(f(N))$
b. $T_1(N) - T_2(N) = o(f(N))$
c. $T_1(N) / T_2(N) = O(1)$
d. $T_1(N) = O(T_2(N))$

4. (10 points) (adapted from Weiss 2.7) Give the Big-Oh for each of the following code excerpts. For parts (a) – (c), verify your Big-Oh doing a precise algorithm analysis, using summations and reducing to closed forms as demonstrated in class. You may want to refer to section 1.2.3 in the book for series formulas. For full credit, show your work of how you used summations to reduce to closed form. For part (d), give a brief explanation as to how you came up with your Big-Oh.

a)

```
sum = 0;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
        sum++;
        sum++;
    }
}
```

b)

```
sum = 0;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= 3 * i; j++) {
        sum++;
    }
    for (k = 1; k <= 100000; k++) {
        sum++;
    }
}
```

c)

```
sum = 0;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= i * i; j++) {
        sum++;
    }
}
```

d)

```
sum = 0;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= i * i; j++) {
        if (j % i == 0) {
            for (k = 1; k <= j; k++) {
                sum++;
            }
        }
    }
}
```

5. (12 points)

(a) Write the following Java method (your code should probably be under twenty lines):

```
public static int firstNonSmallerIndex(int[] array, int value)
```

The method takes in an array in sorted order and a value, and returns the smallest possible index of an element that is equal to or larger than the given value (or -1 if the value is larger than the max).

Your method must run in $O(\log N)$ time provided the list has few duplicates.

Assuming `array = {1,2,3,3,3,4,5,5,14,17},` here are some example calls:

Method call	Return value
`firstNonSmallerIndex(array, 3)`	2
`firstNonSmallerIndex(array, 4)`	5
`firstNonSmallerIndex(array, -1)`	0
`firstNonSmallerIndex(array, 23)`	-1
`firstNonSmallerIndex(array, 15)`	9

*Hint: Your code will look similar to binary search. Once you have found one non-smaller index, it doesn't mean that it is the **first** non-smaller index. In other words, you should keep on searching until you are sure you have found the first non-smaller index.*

(b) After implementing your method in Java and ensuring that it is correct, run timing tests on your method with arrays of different sizes. Use the method `createRandomSortedArray` (shown later) and `System.nanoTime()` (see http://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime%28%29) to help you create random sorted arrays and run your timing tests. Answer the following questions:

- What array sizes did you choose and why?
- What were the runtimes of each array size?
- Did your runtimes increase as you expected according to the Big-Oh of your algorithm? Why or why not?

For part (a), your `firstNonSmallerIndex` method should be found in a class named `FirstNonSmaller` and should be saved in a file named `FirstNonSmaller.java`. Turn in `FirstNonSmaller.java` electronically by submitting it to the turn-in link on the homework webpage.

For part (b), you can save the code/methods you use to do your timing tests in `FirstNonSmaller.java`–we will **not** grade your timing code. However, we will grade your answers to the questions above. Save your answers in a file named `README.txt` and also submit that file using the same turn-in link on the homework webpage.

```
import java.util.*;

....

public static int[] createRandomSortedArray(int size) {
    Random rand = new Random();
    int[] array = new int[size];

    for (int i = 0; i < size; i++) {
        // pick random numbers (subtract a bit so that some
        // are negative)
        array[i] = rand.nextInt(size * 3) - size / 4;
    }

    Arrays.sort(array);

    return array;
}
```