CS 240 Homework 1

Alex Vondrak

Fall 2011

1. Java has certain methods which take a variable number of arguments. For instance, we've seen the `printf` method of the `PrintStream` class takes a single format string, then some number of arguments corresponding to the values that will be inserted into that format string—`printf("No args")`, `printf("One arg: %s", "foo")`, `printf("Two args: %s %s", "foo", "bar")`, etc.

   The way these methods do this is with a *vararg*: if the last parameter in a method declaration is of the form `type... name`, then zero or more "trailing" inputs are collected up into a `type[]` array called `name`. For instance, the signature for `printf` is

   ```
   public PrintStream printf(String format, Object... args)
   ```

   Write a method

   ```
   public static int sumBigger(int limit, int... values)
   ```

   which returns the sum of only those values that are bigger than `limit`. E.g.,

   ```
   sumBigger(100)           // == 0
   sumBigger(0, 1, 2, 3)    // == 6
   sumBigger(1, 1, 2, 3)    // == 5
   sumBigger(100, 1, 2, 3) // == 0
   ```

2. Java has a special **`for`**-loop syntax to cut down on repetitive code. Instead of writing

   ```
   int[] a = ...;
   for (int i = 0; i < a.length; i++) {
     // do something with a[i]
   }
   ```

   you can instead use an *enhanced* **`for`**-loop to iteratively assign a variable to each element of the array, like

   ```
   int[] a = ...;
   for (int n : a) {
     // do something with n
   }
   ```

Write a method

```
public static int count(int element, int[] elements)
```

that uses an enhanced **for**-loop to count the number of times `element` appears in the `elements` array.

3. Rearrange the following sequence of functions so that each is O of the next: n, $\sqrt{n}$, $\log n$, $\log\log n$, $(\log n)^2$, $n/\log n$, $(1/3)^n$, $(3/2)^n$, 17.

4. Prove that O is transitive. I.e., show that $f \in O(g) \wedge g \in O(h) \implies f \in O(h)$.

5. Consider the following code, where `n` is an arbitrary **int**:

```
for(int i = 0; i < n-1; i++) {
   for(int j = i+1; j < n; j++) {
      for(int k = 0; k < j; k++) {
         // any sequence of operations whose
         // worst-case running time is O(1)
      }
   }
}
```

What is this loop's worst-case running time in terms of O of a function of `n`?