Quiz 23: Big-O runtime SOLUTIONS

CSCI 110 Section 1

Wednesday, November 2, 2016

1) What's the Big-O runtime of the following function? [10 points]

```
void mystery(int n) {
    for (int i = 0; i < n; i += 3) {
        System.out.println(i);
    }
}
```

O(n)

i gets incremented by 3, but the runtime is linear because we disregard constant coefficients.

2) What's the Big-O runtime of the following function? [10 points]

```
void mystery(int n) {
    System.out.println("hello world");
}
```

O(1)

always takes the same amount of time no matter the input

3) What's the Big-O runtime of the following function, assuming "arr" has n elements? [10 points]

```
void mystery(ArrayList<Double> arr) {
    for (int i = 0; i < arr.size(); i++) {
        System.out.println(arr.contains(arr.get(i) + 1));
    }
}
```

$O(n^2)$

Each call to contains() takes O(n) time. We call contains() n times. Multiplying, we get $O(n^2)$.

4) What's the Big-O runtime of the following function? [10 points]

```
int mystery(int n) {
    int sum = 0;
    for (int i = 0; i < n; i *= 2) {
        sum += i;
    }
    return sum;
}
```

O(log n)

We have to double n to increase the runtime by a constant amount. (Think about how many iterations of the for loop run for n = 2, n = 4, n = 32, n = 64, etc.) That means the runtime is logarithmic (base 2) when related to n.

5) What's the Big-O runtime of the following function, assuming "arr" has n elements along each dimension? [20 points]

```
void mystery(int[][][][] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[0].length; j++) {
            for (int k = 0; k < arr[0][0].length; k++) {
                for (int m = 0; m < arr[0][0][0].length; m++) {
                    sum += arr[i][j][k][m];
                }
            }
        }
    }
}
```

$O(n^4)$

Arr is a 4-dimensional array of ints where all edges are of length n. We iterate through every element.

6) What's the Big-O runtime of the following function? [20 points]

```
int mystery(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * mystery(n-1);
}
```

O(n)

This is the recursive factorial function. It gets called n times and the base case kicks in when n becomes 1.

7) What's the Big-O runtime of the following function? [20 points]

```
double mystery(double n) {
    if (n < 65.556) {
        return n * 4;
    }
    return mystery(n - 2.0) + mystery(n - 4.0) + mystery(n - 7.0);
}
```

$O(3^n)$

This is similar to recursive fibonacci, but slower. Here, we branch off three calls for every single call to mystery(). And each of those calls invokes the function three times as well, and so on.

8) What's the Big-O runtime of the following function? [extra credit, 10 points]

```
void mystery(int n, int m) {
    int a = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            a += 1;
        }
    }
}
```

O(mn)

Something that takes an amount of work proportional to m gets executed n times.