CS 201 Discussion 5

BIG-OH REVIEW
A step by step approach to finding Big-Oh runtime

You learned about Big-Oh in lecture on Friday. Here’s a summary of how you can find the Big-Oh runtime of a snippet of code.

1. Determine how many steps (assigning variables, adding, multiplying, etc) the code takes as a function of the input. (e.g. if the code is nothing but for loops which iterate a total of $2n^2 + 2n + 1$ times, then that’s our function).

2. Remove all terms but the highest-ordered (e.g. $2n^2 + 2n + 1 \rightarrow 2n^2$)

3. Remove any constant multipliers (e.g. $2n^2 \rightarrow n^2$).

4. Put an $O(\ldots)$ around it ($O(n^2)$)
Big-Oh Practice Problems - 1

For each of the below snippets of code, what is the runtime in terms of n, using Big-Oh notation?

A.

```java
public int calc(int n){
    int sum = 0;
    for(int k = 1; k <= n; k = k * 2){
        sum++;
    }
    return sum;
}
```

B.

```java
public int calc2(int n){
    int sum = 0;
    for(int j = 0; j < n; j++){
        for(int k = 0; k < j; k++){
            sum++;
        }
    }
    return sum;
}
```
Big-Oh Practice Problems - 1

For each of the below snippets of code, what is the runtime in terms of n, using Big-Oh notation?

C.
public int calc3(int n){
    int sum = 0;
    for(int k = 0; k < n; k++){
        sum++;
    }
    for(int k = 0; k < n; k++){
        sum++;
    }
    return sum;
}

D.
public int calc4(int n){
    int prod = 1;
    for(int k=0; k < n; k++){
        for(int j = k; j >= 1; j = j/2){
            prod *= j;
        }
    }
    return prod;
}
Big-Oh Practice Problems - 1

For each of the below snippets of code, what is the runtime in terms of n, using Big-Oh notation?

E.
```java
public int calc5(int n){
    int prod = 1;
    for(int k = n; k >= 1; k = k/2){
        prod *= k;
    }
    for(int k = 0; k < n; k++){
        prod += k;
    }
    return prod;
}
```

F.
```java
public int calc6(int n){
    int p = 0;
    while (p*p < n){
        p++;
    }
    return p;
}
```
Consider the following three algorithms for determining whether anyone in the room has the same birthday as you.

**Algorithm 1**: You say your birthday, and ask whether anyone in the room has the same birthday. If anyone does have the same birthday, they answer yes.

**Algorithm 2**: You tell the first person your birthday, and ask if they have the same birthday; if they say no, you tell the second person your birthday and ask whether they have the same birthday; etc, for each person in the room.

**Algorithm 3**: You only ask questions of person 1, who only asks questions of person 2, who only asks questions of person 3, etc. You tell person 1 your birthday, and ask if they have the same birthday; if they say no, you ask them to find out about person 2. Person 1 asks person 2 and tells you the answer. If it is no, you ask person 1 to find out about person 3. Person 1 asks person 2 to find out about person 3, etc.
Algorithm 1: You say your birthday, and ask whether anyone in the room has the same birthday. If anyone does have the same birthday, they answer yes.

Algorithm 2: You tell the first person your birthday, and ask if they have the same birthday; if they say no, you tell the second person your birthday and ask whether they have the same birthday; etc, for each person in the room.

Algorithm 3: You only ask questions of person 1, who only asks questions of person 2, who only asks questions of person 3, etc. You tell person 1 your birthday, and ask if they have the same birthday; if they say no, you ask them to find out about person 2. Person 1 asks person 2 and tells you the answer. If it is no, you ask person 1 to find out about person 3. Person 1 asks person 2 to find out about person 3, etc.

For each algorithm, what is the factor that can affect the number of questions asked (the "problem size")?

In the worst case, how many questions will be asked for each of the three algorithms?

For each algorithm, say whether it is constant, linear, or quadratic in the problem size in the worst case.
Bill shows you an algorithm to optimally choose classes for all students at Duke. You, being a good Computer Science student, notice that the big-Oh complexity of the algorithm is $O(2^n)$ where $n$ is the number of students.

However, Bill demonstrates the program for a sample of 100 students and it returns the schedules almost immediately. Bill says that his algorithm is good enough for Duke. He mentions something about Moore's Law and states that “Computers are getting faster at an exponential rate. That is, every 18 months, they double in speed. Even if the program is not fast enough now, it will be soon.”

Bill is off a little bit on what Moore's law means. However, given that computers continue doubling in speed every 18 months, will the $O(2^n)$ algorithm ever be practical? Explain why or why not?
public class PointStuff{
    public static class Point{
        int x;
        int y;
        public Point(int x, int y){
            this.x = x;
            this.y = y;
        }
        public double distanceFrom(Point p){
            return Math.sqrt( (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) );
        }
    }
}

public Point[] closestPair(Point[] list){
    double closest = Double.MAX_VALUE;
    Point[] ret = new Point[2];
    for(int j = 0; j < list.length; j++){
        for(int k = j+1; k < list.length; k++){
            if (list[j].distanceFrom(list[k]) < closest){
                ret[0] = list[j];
                ret[1] = list[k];
            }
        }
    }
    return ret;
}
Test Review

At this point, we can answer any last-minute questions you may have regarding content for Midterm 1.