CS330 – Order of Growth with Loops

Purpose: When there are multiple ways to solve a problem, we need to be able to compare them in order to choose the best one. Time efficiency of algorithms is measured by length of time that an algorithm runs as a function of the size of the input. Using this analysis we can compare algorithm efficiency without having to actually measuring the running time.

Knowledge: This activity will help you become familiar with the following content knowledge:

- Determine the proportional running time of code involving loops.
- Use the formal definition of Big-Oh.

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. We will start by counting things.

   Consider the following algorithm:

   ```
   a = f(0)
   for (i = 1; i<=n; i++){
       a = a + f(i)
   }
   ```

   How many calls are made to the function `f`?

2. Consider the following algorithm:

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

   How many calls are made to the function `f`?

3. Let's change the algorithm above so that the `j` loop starts at `i` rather than 1:

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

   How many calls are made to the function `f`?

   Hint: $1 + 2 + ... + n = \frac{n(n+1)}{2}$

4. Consider this one:

```
a = f(0)
for (i = 1; i<=n; i++){
    for (j = 1; j<=n; j++){
          for (k = j; k<=n; k++){
               a = a + f(i) + j + k
          }
    }
}
```

   How many calls are made to the function `f`?

5. The count of how many times things occur gives us a sense of the running time of the algorithm. We want to examine what happens as the problem size increases.

   For example, consider two algorithms A and B. Algorithm A takes $3n^3$ nanoseconds to process input of size n versus algorithm B that takes $20,000,000n$ nanoseconds to process input of size n. Be aware that a nanosecond is 10^{-9} seconds, a microsecond is 10^{-6} seconds and a millisecond is 10^{-3} seconds.

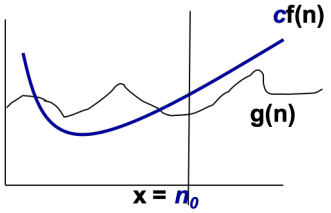
   If $n = 10$ then A takes 3 microseconds and B 200 milliseconds.
   If $n = 100$ then A takes 3 microseconds and B 2 seconds.
   If $n = 1000$ then A takes 3 seconds and B 20 seconds.

   How much time would each algorithm take for $n = 1,000,000$? (Represent the time in hours, days, or even years if appropriate to get the sense of scale).

6. We use Big Oh to describe the rate of growth of an algorithm. The intuition for Big Oh is that we can avoid details when they don't matter and they don't matter when the problem size n is big enough.

   What do these functions have in common?
   n^2
   $.001n^2$
   $1000n^2$
   $5n^2 + 3n + 2\log n$

7. A more formal definition of Big Oh is that it is an upper bound. Formally, $g(n)$ is $O(f(n))$ if there exists constants c and n_0 such that $g(n) < cf(n)$ for all $n > n_0$.



   Give possible c and n_0 to show that $T(n) = n^2 + 2n + 1$ is $O(n^2)$. (There are many possible values that will work but justify your choices.)

8. Give possible c and n_0 to show that $T(n) = 2^n + 5n$ is $O(2^n)$.

9. We can use a proof by contradiction to show that $T(n) = 2n^2$ is not $O(n)$. In a proof by contradiction we assume that there exists an n_0 and c such that $2n^2 < cn$ for all $n > n_0$. Why does this assumption lead to a contradiction?

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies.

Assignment 1:
Give the computational complexity of each of the following pieces of code in Big-Oh notation and explain how you arrived at your result.

1.
```
for (i=1; i<=n; i=i+2){
   // constant number of operations
 }
```

2.
```
for (i=1; i<=n; i++){
  for (j=i; j<=n; j=j+2){
   // constant number of operations
  }
 }
```

3.
```
for (j=n; j>1; j=j/2){
   // constant number of operations
}
```

4.
```
for (i=1; i<=n; i++){
  for (j=n; j>1; j=j/2){
   // constant number of operations
  }
 }
```

Criteria for Success: You have a Big-Oh and a brief explanation for each code fragment.

Assignment 2:
Use the formal definition of Big-Oh to answer each of the following questions.

1. Is $n^2 + 100$ of complexity $O(n^2)$?
2. Is $n^2 + 100$ of complexity $O(n^3)$?
3. Is 2^{n+1} of complexity $O(2^n)$?
4. Is 2^{2n} of complexity $O(2^n)$?

Criteria for Success: You need to provide the c and n_0 values for the formal definition of Big-Oh or show that no c and n_0 values exist by using a proof by contradiction.