COMP161 - Lab 10

Spring 2016

Abstract

In this lab you'll work some basic exercises on Big-O and algorithm analysis. Write solutions on this sheet and bring the sheet to class tomorrow. We'll discuss it, then you'll hand it in.

1. Rank the following complexity from least to greatest order, in terms of resources needs by members of the class, where 1 is the least resource intense class and numbers proceed up from there.

$O(n \log n)$ _____

$O(n^2)$ _____

$O(1)$ _____

$O(2^n)$ _____

$O(n^3)$ _____

$O(\log n)$ _____

$O(n)$ _____

2. Determine the Big-O for each of the following functions.

   (a) $3n + 1500$

   (b) $\frac{x \log x}{25} - 5x + 3x^3$

   (c) $(n + 1)(n + 2)$

   (d) 10^9

   (e) $\log y + 135 + \frac{y}{2}$

3. A lot of looped/repeated code generates a pattern of incrementally increasing work. For example, the first iteration carries out 1 operation, then the next 2, the next 3, and so on up to some n. This pattern, called the *arithmetic series*, is well studied in mathematics and is easily expressed with *summation notation*.

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + (n-1) + n = \frac{n(n+1)}{2}$$

For now, the import thing to notice is that we can determine the total of the sum without explicitly doing all the the addition by using the formula on the right in the above equation. What's the Big-O of this summation?

4. Consider the following loop that counts through $[0, n)$:

```
for(int i{0} ; i < n ; ++i){
}
```

If the initialization of i costs one operation, the $<$ costs one operation each time it executes, and $++$ costs 2 operators (because it's the same as $i = i + 1$) each time it executes, then what's the total cost of this loop and what's the Big-O? [Double check your answer before moving on from this question]

5. Rewrite the loop from the previous question so that it counts through only the even numbers. Do this without doing anything inside the loop. To check yourself, write the loop in a main procedure so that you can pick an n and print the numbers like this:

```
int n{ ... }; // pick a number for n, not too big, not too small
for(int i{0}; ... ; ... ){
   std::cout << i << '\n';
}
```

Intuitively we'd expect this to do half as much work as the original loop, what's that mean in Big-O terms? To check your intuition determine the exact number of operations needed for the loop and then Big-O of that.

6. Repeat the previous exercise but write a loop that counts by 5, i.e. counts through every fifth number in $[0, n)$.

7. What do the previous exercises tell you about the effect of taking fixed sized steps with a counting loop?

8. Consider the following nested loop structure:

```
for(int i{0}; i < n; i++){
  for(int j{0}; j < n; j++){
  }
}
```

Ignoring the interaction with the other loop, determine the exact number of operations and Big-O of each loop. Now, what's the exact operations done by the whole thing and what's the Big-O of that? To better understand the execution of these loops, code this up to print i and j like this:

```
int n{ ... };
std::cout << "i js\n";
for( int i{0}; i < n ; ++i){
   std::cout << i << " ";
   for( int j{0}; j < m; ++j){
      std::cout << j << ' ';
   }
   std::cout << "\n";
}
```

9. Consider the following nested loop structure:

```
for(int i{0}; i < n; i++){
  for(int j{i}; j < n; j++){
  }
}
```

Code these loops up in the same fashion as the previous loop and see what they do. What's different about this code compared to the previous nested loop? Will it do more, less, or the same amount of work as the previous code? Now... if $n = 5$, how many times is the inner loop executed and how many total operations does it carry out on each of those execution? What's the Big-O of this code? Why?

10. Would changing the step size of either loop in either of the above two examples change the complexity of the code?

11. These loops tell us something about the efficiency limitations of our iterative strategy, what is it?