**CSCI-2270**

Lab 3

Instructor: Boese

**Algorithm Analysis**

**Objectives**

* Reinforce your understanding of algorithm analysis

Write down your answers (or print this out and bring to lab to fill out).

The TA will walk you through a discussion of each.

**True/False**

1. True or False – In terms of efficiency, all algorithms are equivalent.

False

1. True or False – Algorithm analysis uses a mathematical notation called Big-O to analyze any given algorithm.

False

1. True or False – Big-O is really the Greek symbol Theta.

False

1. True or False – What really determines how efficient an algorithm is is the processor speed of the computer on which it is running.

False

1. True or False – Given a SEQUENTIAL algorithm (no branches or loops) with ten lines of code, Θ= 10c, where c is some machine dependent constant we won’t care about.

False

**Calculations**

1. If you had an algorithm (developed by some twisted person!) that included THREE nested loops, what would the algorithm complexity be?
   1. 3n
   2. n3
   3. 3n + 30
   4. There is no way of knowing
2. One of the most frequently deployed algorithms in Computer Science applications is a Search. Which search algorithm looks at each element in a list until it finds a match? For example, you have a key chain with 50 keys and you need to find a specific key to unlock a door…
   1. Binary
   2. Random
   3. Sequential
   4. Quick
3. On average (ie, you run your key search 100 times), how long will it take you to locate a match using a Sequential Search?
   1. 1 time
   2. n times
   3. 2n times
   4. n/2 times
4. What is the worst case scenario on how long it will take you to locate a match using a Sequential Search?

n times

1. What are the pre-conditions of when you can use a Binary Search on a list?

Sorted array

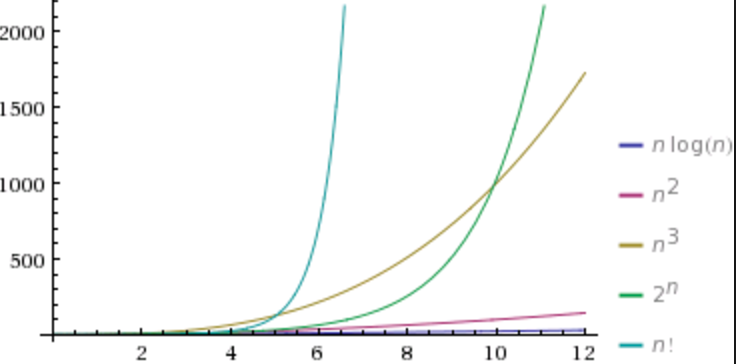
Know number of items

Constant O(1) access time

1. Why do we drop the constants from Big-O notation?

Care about classes of algorithms and asymptotic behavior as n 🡪 infinity not so much about the scaling of n from the constant

1. What happens when the Big-O of an algorithm is more efficient for smaller values of n but less efficient for large values of n? *(e.g.,* O(2n) *vs* O(n3) *)*



Care about asymptotic behavior as n 🡪 infinity. We don’t care about small values of n for big O

**To get credit for this lab exercise, show the TA and sign the lab’s completion log.**