HW 3

Quinn Schiller

April 8, 2016

3.1

1. Using the definition of Big O, show that:

   (a) $6n^2 + 3$ is $O(n^2)$

   let $n \geq 1$.
   $6n^2 \leq 6n^2 + 3$
   $6n^2 \leq 7n^2$
   $c = 7$
   Therefore $6n^2 + 3$ is $O(n^2)$

   (b) $n^2 + 17n + 1$ is $O(n^2)$

   let $n \geq 1$.
   $n^2 + 17n + 1 \leq n^2 + 18n$
   $n^2 + 18n^2 \leq n^2 + 18n$
   $19n^2 \leq 20n^2$
   $c = 20$
   Therefore $n^2 + 17n + 1$ is $O(n^2)$

   (c) $5n^3 + 100n^2 - n - 10$ is $O(n^3)$

   let $n \geq 1$.
   $5n^3 + 100n^2 - n - 10 \leq 5n^3 + 100n^2$
   $5n^3 + 100n^2 \leq 105n^3$
   $c = 105$
   Therefore $5n^3 + 100n^2 - n - 10$ is $O(n^3)$

   (d) $3n^2 + 2^n$ is $O(2^n)$

   let $n \geq 3$.
   $3n^2 + 2^n \leq 2(2^n)$
   $c = 2$
   Therefore $3n^2 + 2^n$ is $O(2^n)$

2. Show that $7n^2 + 5n$ is not $O(n)$.

   No matter how high the c value is set, $7n^2 + 5n$ will eventually pass cn. $7n^2$ can be thought of as $7n \times n$. The O(n) function is a $c \times n$. By factoring out the n term, we get $n(7n) = n(c)$. Eventually, 7n will be larger than c because 7n grows and c does not. Therefore $7n^2 + 5n \nleq O(n)$ and is therefore not correct.

3. Consider four programsA, B, C, and D that have the following performances: If each program requires 10 seconds to solve a problem of size 1000, estimate the time required by each program for a problem of size 2000.

   (a) O(log n)

   $c \log 1000 = 10$
   $c = 10/3$
   $10/3 \log 2000 = 11$ seconds

   (b) O(n)

   $c1000 = 10$
   $c = 1/100$
   $1/100 \times 2000 = 20$ seconds

   (c) O(n^2)

   $c1000^2 = 10$
   $c = 1/100000$
   $1/100000 \times 2000^2 = 40$ seconds

   (d) O(2^n)

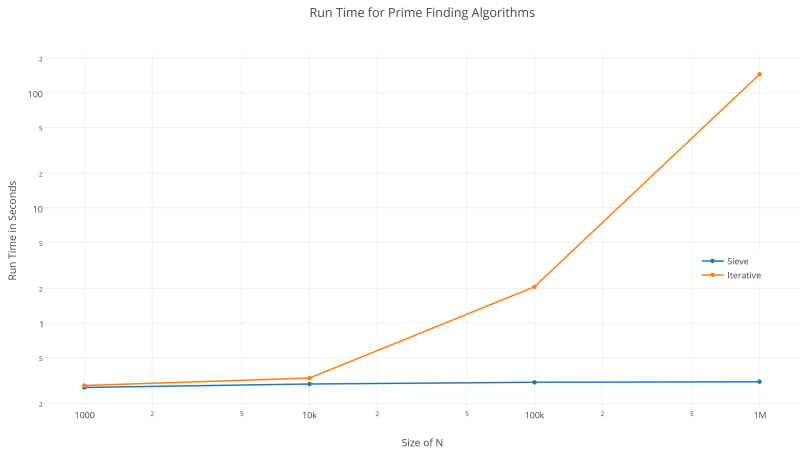
   $c2^{1000} = 10$
   c = REALLLY small
   REALLY small 2^{2000} = An unbelievably huge number.

3.2

1. Run your code for the following values of N: 1000, 10000, 100000, 1000000 on both algorithms. Time your results and create a chart. Turn off output when you are timing your results.

   - With the exception of the 1000000 Iterative test, these times are the average of 100 trials.

Size	Sieve Time	Iterative Time
1000	0.27539 seconds	0.28657 seconds
10000	0.29495 seconds	0.33225 seconds
100000	0.30516 seconds	2.06104 seconds
1000000	0.30940 seconds	145.011 seconds



2. What is the Big-O time for the *isPrime* algorithm? Determining the Big-O time for *sieve* is nontrivial. What would you guess that it is closest to, of the standard growth functions O(log n), O(n), O(n log n), O(n^2) ?

   - The Big-O time for *isPrime* pretty simple. It is two nested loops. The outer loop runs n times. At it's worst case, the inner loop runs n times. Together they are $O(n^2)$.
   - The Big-O time for *sieve* is probably closest to $n \log n$ because outer loop runs n times and the inner loop probably grows at a logarithmic rate because primes become more rare as the numbers get larger

3. The *isPrime* method could be improved by having its loop run from 2 to ($n^{.5}$) instead if 2 to n. How would that change its Big-O time?

   - The outside loop still runs n times. The inside loop now runs the square root as many times so it is now $n(\sqrt{n})$ long or $O(n^{1.5})$.