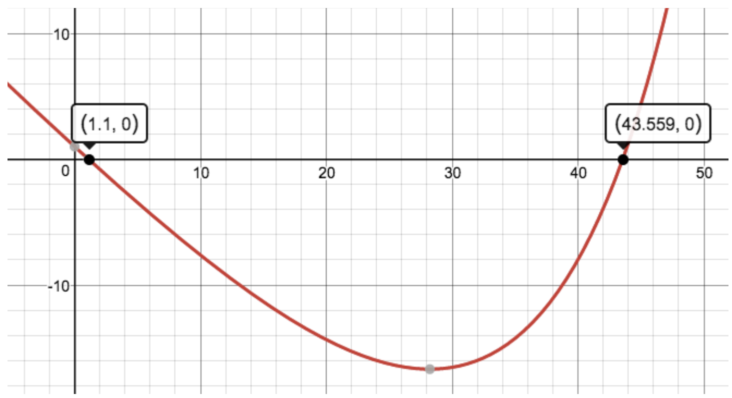
Kelsey Helms
CS 325
9/28/16
Homework 1

1. **Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64\, n \lg n$ steps. For which values of n does insertion sort beat merge sort?**

$$8n^2 < 64\, n \lg n$$
$$n < 8 \lg n$$
$$\frac{n}{8} < \lg n$$
$$2^{\frac{n}{8}} < n$$
$$2^{\frac{n}{8}} - n < 0$$



By graphing this equation, we can see that $2^{\frac{n}{8}} - n < 0$ from the interval $1.1 < n < 43.5$. Therefore, insertion sort is faster from $1.1 < n < 43.5$

2. **Fill in the given table. Hint: It may be helpful to use a spreadsheet or Wolfram Alpha to find the values. *Using 30 days for month and leap years for century calculation.***

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{1E6}	2^{6E7}	$2^{3.6E9}$	$2^{8.64E10}$	$2^{2.592E12}$	$2^{3.1536E13}$	$2^{3.1556E15}$
$\sqrt{n}$	$1E^{12}$	3.6E15	1.296E19	7.4649E21	6.7184E24	9.9451E26	9.9451E30
n	1E6	6E7	3.6E9	8.64E10	2.592E12	3.1536E13	3.1556E15
$n \lg n$	6.2746E4	2.8014E6	1.3337E8	2.7551E9	7.187E10	7.9763E11	6.8656E13
n^2	1E3	7.7459E3	6E4	2.9393E5	1.6099E6	5.6156E6	5.7596E7
n^3	100	391	1532	4420	13736	31593	159397
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

3. **Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence**

$$\boldsymbol{T(n) = \begin{cases} 2, & if\ n = 2 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n = 2^k, for\ k > 1 \end{cases}}$$

is $T(n) = n \lg n$

Base Step: If n = 2, then T(2) = 2 lg 2 = 2.
Inductive Hypothesis: Assume T(n) = n lg n is true if n = 2^k for some integer k > 1.
Axiom of Induction: We must prove that $T(2^{k+1}) = 2^{k+1} \log 2^{k+1}$. If n = 2^{k+1}, then

$$T(2^{k+1}) = 2T\left(\frac{2^{k+1}}{2}\right) + 2^{k+1}$$
$$= 2T(2^k) + 2^{k+1}$$
$$= 2(2^k \log 2^k) + 2^{k+1}$$
$$= 2^{k+1}((\log 2^k) + 1)$$
$$= 2^{k+1} \log 2^{k+1}$$

which is what we were trying to prove.

4. For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

a. f(n) = $n^{0.75}$; g(n) = $n^{0.5}$
f(n) is Ω(g(n)) because $n^{0.75}$ increases faster than $n^{0.5}$

b. f(n) = n; g(n) = $\log^2$ n
f(n) is Ω(g(n)) because n increases faster than log2 n.

c. f(n) = log n; g(n) = lg n
f(n) is Θ(g(n)) because the base is insignificant.

d. f(n) = e^n; g(n) = 2^n
f(n) is Ω(g(n)) because e^n increases faster than 2^n

e. f(n) = 2^n; g(n) = 2^{n-1}
f(n) is Θ(g(n)) because the -1 disappears as a constant

f. f(n) = 2^n; g(n) =2^{2^n}
f(n) is O(g(n)) because 2^{2^n} increases faster than 2^n

g. f(n) = 2^n; g(n) = n!
f(n) is O(g(n)) because n! increases faster than 2^n

h. f(n) = nlgn; g(n) = n√n
f(n) is O(g(n)) because n√n increases faster than nlgn.

5. **Design an algorithm that given a list of n numbers, returns the largest and smallest numbers in the list. How many comparisons does your algorithm do in the worst case? Instead of asymptotic behavior suppose we are concerned about the coefficients of the running time, can you design an algorithm that performs at most 1.5n comparisons? Demonstrate the execution of the algorithm with the input A= [9, 3, 5, 10, 1, 7, 12].**

Pseudocode:

```
for 0 to n/2
        if x_i < x_i+1
                x_i goes to array small
                x_i+1 goes to array large
        else
                x_i goes to array large
                x_i+1 goes to array small
if A is not empy
        remaining is smallest
else if x_1 < x_2 in array small
        smallest is x_1
else
        smallest is x_2
if A is not empty
        remaining is largest
else if x_1 >x_2 in array large
        largest is x_1
else
        largest is x_2
for 0 to n/2 – 2
        if x_i in array small < smallest
                x_i is smallest
        if x_i in array large > largest
                x_i is largest
return largest and smallest
```

n/2 comparisons are made in the first for loop. Then two comparisons are made in the middle section. Then 2 * (n/2 - 2) comparisons are made in the second for loop. This ends up being $\frac{n}{-}2 + 2 + 2 * \left(\frac{n}{2} - 2\right) = \frac{n}{2} + 2 + n - 4 = \frac{3n}{2} - 2$ which is 2 less comparisons than 1.5n. To illustrate the input A= [9, 3, 5, 10, 1, 7, 12]:

- A[9, 3, 5, 10, 1, 7, 12], small[], large[]
- A[5, 10, 1, 7, 12], small[3], large[9]
- A[1, 7, 12], small[3, 5], large[9, 10]
- A[12], small[3, 5, 1], large[9, 10, 7]

- smallest = 12, largest = 12, small[3, 5, 1], large[9, 10, 7]
- smallest = 3, largest = 12, small[3, 5, 1], large[9, 10, 7]
- smallest = 3, largest = 12, small[3, 5, 1], large[9, 10, 7]
- smallest = 1, largest = 12, small[3, 5, 1], large[9, 10, 7]
- return 1 and 12

6. Let f1 and f2 be asymptotically positive functions. Prove or disprove each of the following conjectures. To disprove give a counter example.

a. $If\ f_1(n) = O(g_1(n))\ and\ f_2(n) = O(g_2(n))\ then\ f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

By definition $f_1(n) = O(g_1(n))$ implies there exist positive constants c_1 and n_0 such that $0 \leq f_1(n) \leq c_1 g_1(n)$ for all $n \geq n_0$. By definition $f_2(n) = O(g_2(n))$ implies there exist positive constants c_2 and n_1 such that $0 \leq f_2(n) \leq c_2 g_2(n)$ for all $n \geq n_1$. Show that $f_1(n) + f_2(n) \leq c_3 (g_1(n)+ g_2(n))$ for all $n \geq n2$.

$$f1(n) + f2(n) \leq c1g1(n) + c2g2(n)$$
$$\leq max(c1, c2)g1(n) + max(c1, c2)g2(n)$$
$$\leq max(c1, c2)\ [g1(n) + g2(n)]$$
$$= c3\ [g1(n) + g2(n)]$$

We've found a c3 = max(c1, c2) that satisfies the definition of big-Oh, which is what we were trying to prove.

b. $If\ f_1(n) = O\big(g_1(n)\big)\ and\ f_2(n) = O\big(g_2(n)\big),\ then\ \frac{f_1(n)}{f_2(n)} = O\left(\frac{g_1(n)}{g_2(n)}\right)$.

Counterexample:

Let $f_1(n) = n$, $f_2(n) = n^2$, $g_1(n) = 3n^2$, $g_2(n) = n^2$. Then $\frac{n}{n^2} = \frac{1}{n}$ which doesn't equal $O(\frac{3n^2}{n^2}) = 1$.

7. Fibonacci Numbers:
The Fibonacci sequence is given by : 0, 1, 1, 2, 3, 5, 8, 13, 21, ….. By definition the Fibonacci sequence starts at 0 and 1 and each subsequent number is the sum of the previous two. In mathematical terms, the sequence F_n of Fibonacci number is defined by the recurrence relation

$$\boldsymbol{Fn = Fn - 1 + Fn - 2\ with\ F0 = 0\ and\ F1 = 1}$$

An algorithm for calculating the nth Fibonacci number can be implemented either recursively or iteratively.

a. **Implement both recursive and iterative algorithms to calculate Fibonacci Numbers in the programming language of your choice. Provide a copy of your code with your HW pdf. We will not be executing the code for this assignment. You are not required to use the flip server for this assignment.**

In Python:

Iterative:

```
def iter_fib(n):
  cur_fib = 0
    prev_fib = 1
   temp_fib = 0
   for i in range(1, n):
       temp_fib = cur_fib + prev_fib
       prev_fib = cur_fib
       cur_fib = temp_fib
   return cur_fib
```

Recursive:

```
def fib(n):
   if n == 0:
      return 0
  elif n == 1:
       return 1
   else:
      return fib(n-1) + fib(n-2)
```

b. **Use the system clock to record the running times of each algorithm for n = 5, 10, 15, 20, 30, 50, 100,1000, 2000, 5000, 10,000, …. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data. If you program in C your algorithm will run faster than if you use python. The goal of this exercise is to collect run time data. You will have to adjust the values of n so that you get times greater than 0.**

Due to the vastly different speeds, the values of n drastically vary between the iterative and recursive versions.

Iterative:

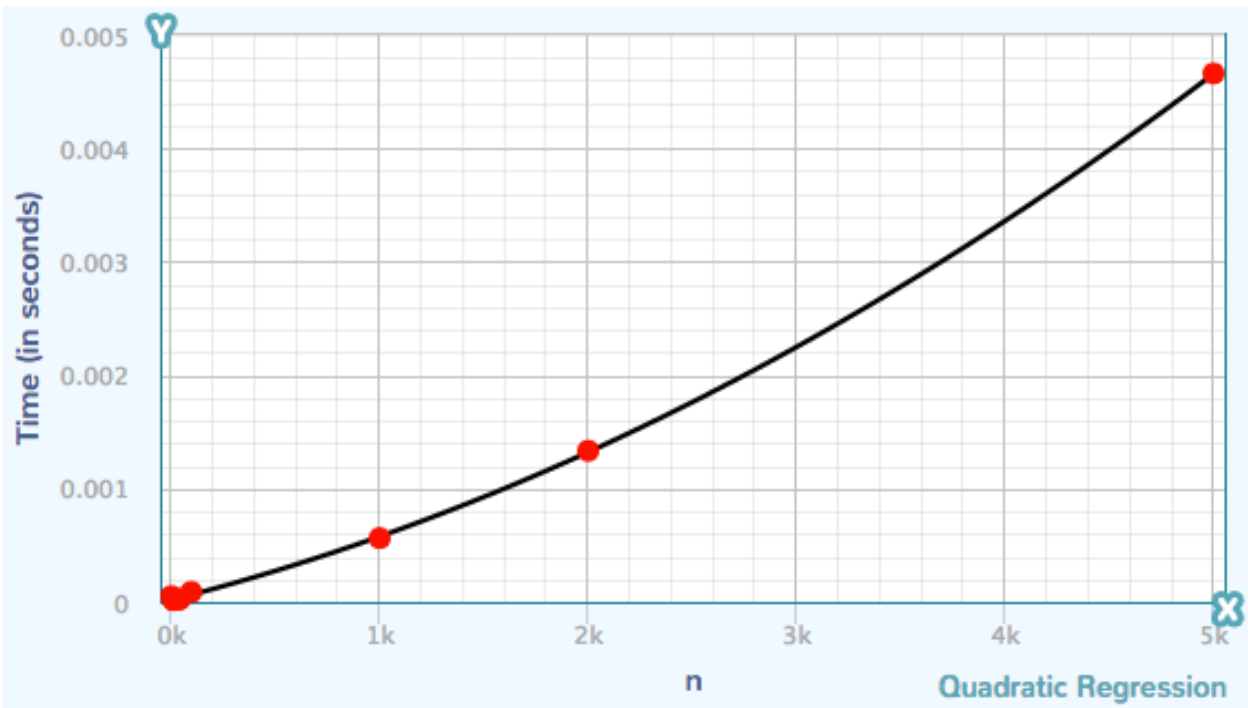
n	Time (in seconds)
5	0.000051021
10	0.000017881
15	0.00001502
30	0.000019788
50	0.000025987
100	0.000087976
1000	0.0005629
2000	0.0013339
5000	.0046608

Recursive:

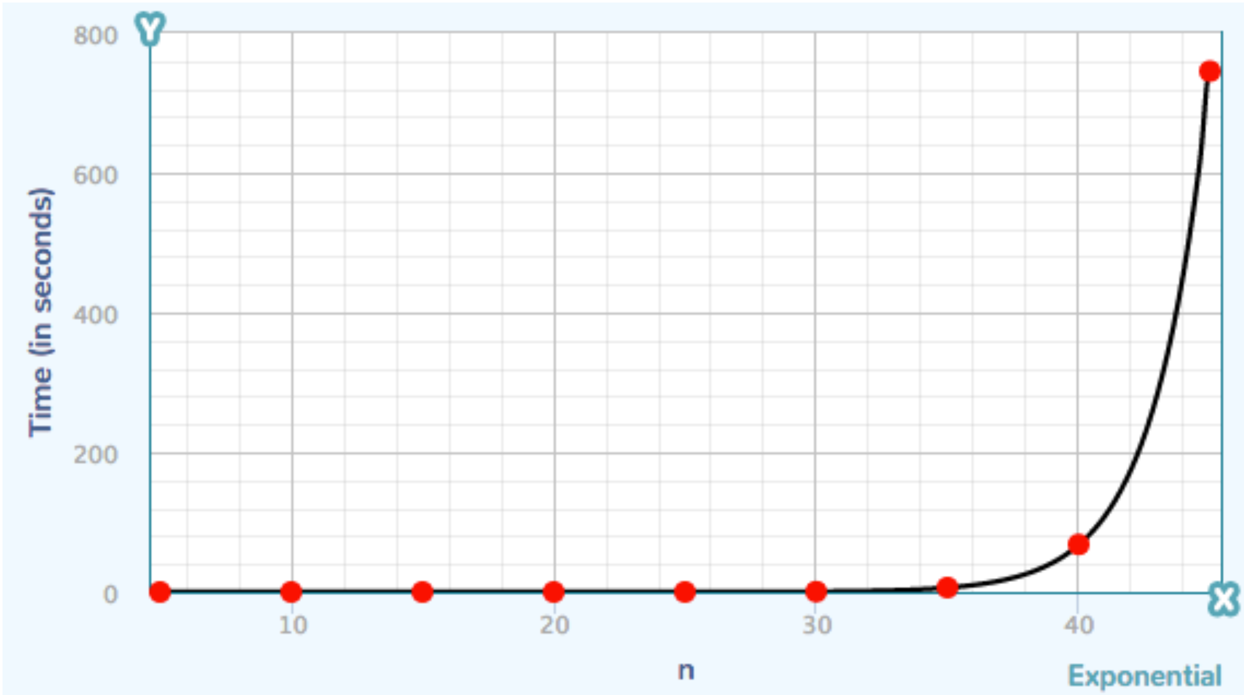
n	Time (in seconds)
5	0.000064849
10	0.000061035
15	0.00038695
20	0.0067908
25	0.046588
30	0.54912
35	6.36967
40	68.10239
45	745.33580

c. **Plot the running time data you collected on graphs with n on the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software.**

Iterative:



Recursive:



d. What type of function (curve) best fits each data set? Again you can use Excel, Matlab, any software or a graphing calculator to calculate the regression curve. Give the equation of the function that best "fits" the data and draw that curve on the data plot. Why is there a difference in running times?

Iterative:
The iterative version had a 2nd degree polynomial best fit, with a .98 accuracy using equation $y = 0.00001847703 + 4.686704e - 7 * x + 9.199769e - 11 * x^2$

Recursive:
The recursive version had an exponential best fit, with a .99 accuracy using the equation $y = 0.01433543 - (-1.585365e - 7/-0.4785349) * (1 - e\text{^}(+0.4785349 * x))$

The recursive solution is much slower because each x_{i-1} is calculated for +1 more times than x_i, creating much more work. When drawn out, the solution forms a binary tree with a depth of n-1. Iteratively, the solution is nearly linear, but with some background work it is a polynomial.