Student: Joaquin Saldana

CS325 Summer 2017

Homework 1

1. Insertion sort runs in $8n^2$ steps, while merge sort runs in $64nlgn$ steps. For what values of n does insertion sort run faster than merge sort?

Insertion sort will run faster than merge sort so long as n is less than or equal to 43

2. The following are the relationships (while dismissing most of the constants):
   a. f(n) = Ω(g(n))
   b. f(n) = Ω(g(n))
   c. f(n) = O(g(n)), logs are all the treated the same
   d. f(n) = Ω(g(n))
   e. f(n) = Ω(g(n))
   f. f(n) = Ω(g(n))
   g. f(n) = Ω(g(n))
   h. f(n) = O(g(n))
   i. f(n) = O(g(n))
   j. f(n) = Ω(g(n))

3. Prove or disprove each of the following conjectures:
   a. If f_1(n) = O(g(n)) and f_2(n) = O(g(n)) then f_1(n) = Θ(f_2(n))
   f1 = O(g(n)) and f2 = O(g(n)) both indicate a upper bound while Θ(g(n)) indicates a tighter bound between two constants. An example is f1 = n^2 + n and f2 = n^. They will have the same O(n^2) upper bound but their constants and lower order terms are different. Therefore f_1(n) ≠ Θ(f_2(n))

   b. f_1(n) ≤ $c_1 g_1(n)$ and f_2(n) ≤ $c_2 g_2(n)$ thus,

$$\begin{aligned} f_1(n) + f_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &\leq c_1 \max\big(g_1(n), g_2(n)\big) + c_2 \max\big(g_1(n), g_2(n)\big) \\ &\leq (c_1 + c_2) \max\big(g_1(n), g_2(n)\big) \end{aligned}$$

   As stated in the asymptotic notation, Big Oh O notation asymptotic is “less than”

4. Submitted via TEACH

5. Modify the insertsort.py and mergesort.py so that it can record the runtime of the algorithm and also generate random integer arrays (with values between 0 and 10,000) of n size. Below are the results:

a. Below are excerpts of the modifications performed to the system

Modfications to Insertion Sort program (the algorithm stayed the same):

```
def main():

        randomIntArray = []

        # what will be the size of the array, this will contain the values 1000, 2000, 5000,
and 10,000
        n = 10000

        while len(randomIntArray) < n:
                randomIntArray.append(randint(0, 10000))

        # will start the timer now that we are about to enter the insertion sort
algorithm/function
        start_time = time.time()

        #insertion sort algorithm
        newArray = insert_sort(randomIntArray)

        # to identify how many seconds have passed we substract from the
        # current time the start time
        print("--- Algorithm took %s seconds --- " % (time.time() - start_time))

        # print newArray

main()

#-------------------------------------------------------------------------------------------
```

Modifications to the Mergesort program (the algorithm stayed the same):

```
def main():

        randomIntArray = []

        # what will be the size of the array, this will contain the values 1000, 2000, 5000,
and 10,000
        n = 10000

        while len(randomIntArray) < n:
                randomIntArray.append(randint(0, 10000))
```

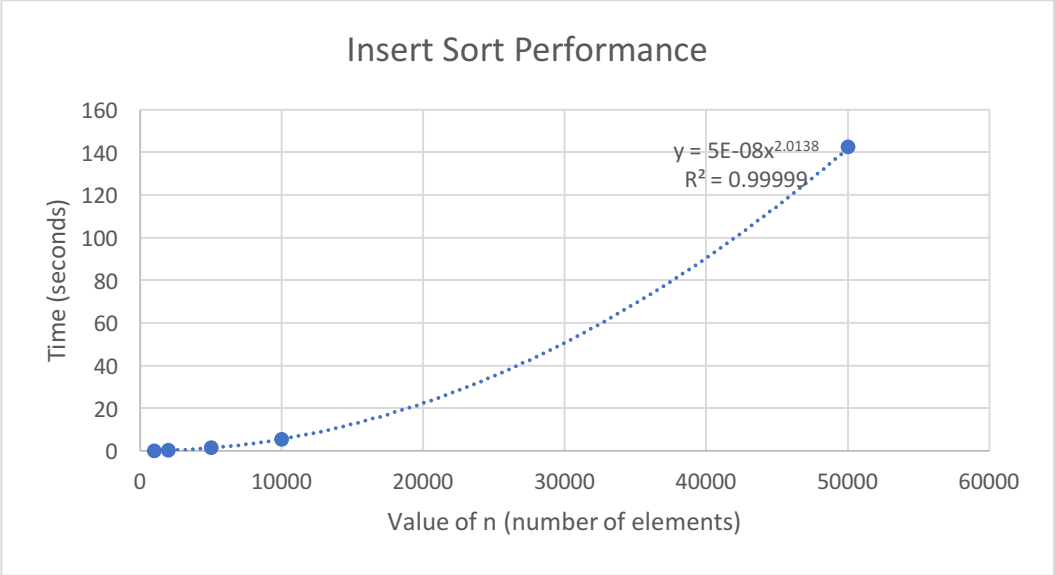
```
        # will start the timer now that we are about to enter the insertion sort
algorithm/function
        start_time = time.time()

        #insertion sort algorithm
        newArray = merge_sort(randomIntArray)

        # to identify how many seconds have passed we substract from the
        # current time the start time
        print("--- Algorithm took %s seconds --- " % (time.time() - start_time))

        print newArray

main()
```

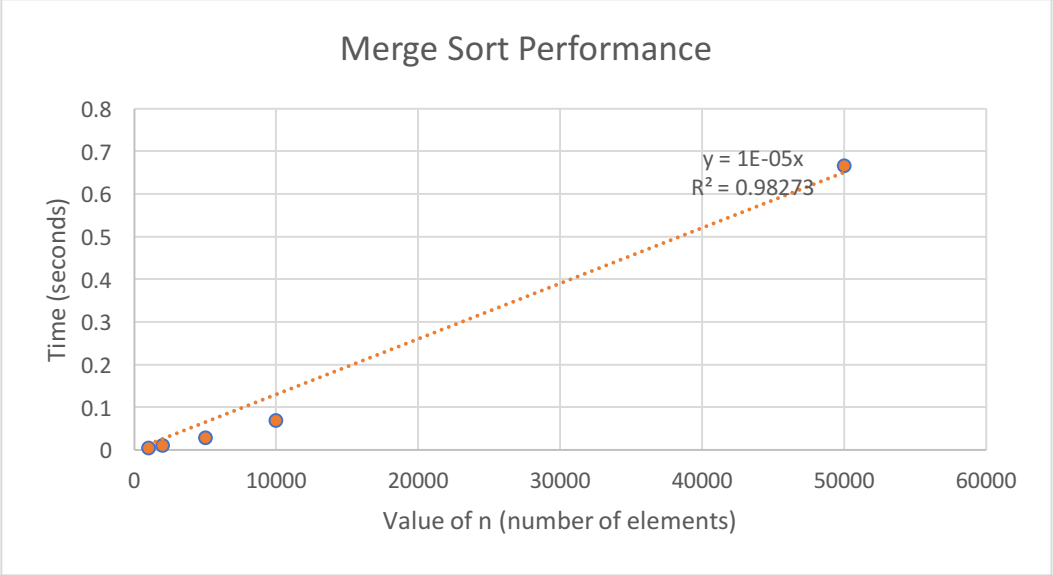
b. Included code to time within the system clock to determine the runtime in seconds. See code above. Used the time.time() function in python.

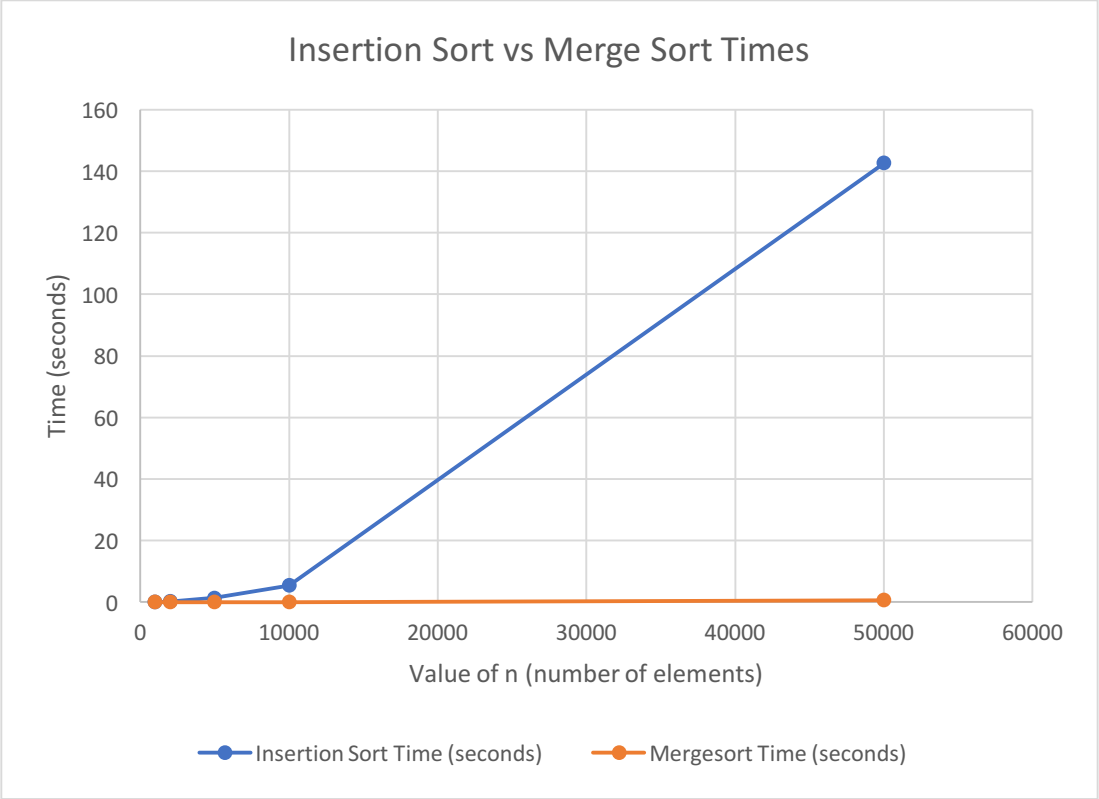
Below is a table which shows the results of our run times:

n	Insertion Sort Time (seconds)	Mergesort Time (seconds)
1000	0.054157019	0.004658937
2000	0.215058088	0.010056019
5000	1.387410164	0.028967857
10000	5.472258091	0.068532944
50000	142.6401329	0.666697979

c. Below are graphs in which the points are plotted on a x-axis and y-axis.


Insert Sort Performance
Time (seconds)
Value of n (number of elements)
y = 5E-08x^2.0138
R² = 0.99999
0
20
40
60
80
100
120
140
160
0
10000
20000
30000
40000
50000
60000


Merge Sort Performance
Time (seconds)
Value of n (number of elements)
y = 1E-05x
R² = 0.98273
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10000
20000
30000
40000
50000
60000



In reviewing the graphs, I can see that insertion sort algorithm is a polynomial algorithm vs. the merge sort which appears more linear. Clearly the merge sort algorithm out performed the insertion sort algorithm. Also, as expected, merge sort performed better than insertion sort when the value of n grew larger.