Christopher Matian
CS325 - Spring 2019
04/07/2019

CS325 - Homework 1

1 -

A) $\lim_{n \to \infty} \frac{n^{0.25}}{n^{0.5}} = \frac{n^{0.25}}{\sqrt{n}} = \frac{1}{n^{.5-.25}} = \frac{1}{n^{.25}}$

infinity property $\lim_{n \to \infty} \left(\frac{1}{n^{.25}} \right) = 0$ $\quad O(g(n))$

B) $\lim_{n \to \infty} \left(\frac{\log n^2}{\log_e n} \right) = \frac{\log_2 n^2}{\ln(n)} = \frac{2 \log_2 n}{\ln(n)} = 2 \cdot \lim_{n \to \infty} \left(\frac{\log_2 n}{\ln(n)} \right)$

$= 2 \cdot \lim_{n \to \infty} \left(\frac{1}{n \ln(2)} \div \frac{1}{x} \right) = 2 \cdot \frac{1}{\ln(2)} = \frac{2}{\ln(2)} > 0 \rightarrow \Theta(g(n))$

C) $\frac{n \log_2 n + n^2}{n \cdot \sqrt{n}} = \frac{n^2 \left(\frac{\log_2 n}{n} + 1 \right)}{n \cdot \sqrt{n}} = \frac{n \left(\frac{\log_2 n}{n} + 1 \right)}{\sqrt{n}}$

$= \sqrt{n} \left(\frac{\log_2 n}{n} + 1 \right) = \infty \cdot 1 = \infty$ $\quad \Omega(g(n))$

D) $\frac{e^n}{2^n} = \left(\frac{e}{2} \right)^n = e^{n \ln\left(\frac{e}{2}\right)} = \infty$ $\quad \Omega(g(n))$

E) $\frac{2^n}{2^{n+1}} = \frac{1}{2^{n+1-n}} = \frac{1}{2} > 0$ $\quad \Theta(g(n))$

F) $\frac{n^n}{n!} = \frac{n^n}{n!}$ $n!$ Factors $= 1 \rightarrow \infty$ $\quad \Omega(g(n))$

2 -

A) $f_1(n) = c_1\, g(n) \quad c_1 > 0$

$f_2(n) = c_2\,(g(n)) \quad c_2 > 0$

$f_1(n) = c_1\, f_2(n) \quad , c_2$

$f_1(n) = c_1\, c_2\, f_2(n)$

$f_1(n) = c_3\, f_2(n)$

$f_1(n) = \Theta(f_2(n))$

B) Big-O means n can grow (the order of N can grow)

To disprove:

$f_1(n) = \frac{1}{n} \leq 1$ for $\forall\ n > 0$

$f_2(n) = \frac{1}{n^2} \leq 1$ for $\forall\ n > 0$

$\frac{f_1(n)}{f_2(n)} = \frac{1}{n} \div \frac{1}{n^2} = n$

$\frac{g_1(n)}{g_2(n)}$ when $n = 1$ is $\frac{1}{1} = 1$

$n \neq 1$

So $\frac{f_1(n)}{f_2(n)} \neq \frac{g_1(n)}{g_2(n)}$

4a -

Merge Sort run time modification:

```
import time
import random
def mergesort(array):
  length = len(array)
  # Array only contains one value
  if (length < 2):
     return array
  # Set the middle of the array and split the array into two based on the midpoint (left and right)
  mid = length // 2
  left = mergesort(array[:mid])
  right = mergesort(array[mid:])
  return merge(left, right)

def merge(left, right):
  result = []
  i = j = 0
  while (i < len(left) and j < len(right)):
     if (left[i] < right[j]):
        result.append(left[i])
        i += 1
     else:
        result.append(right[j])
        j += 1
  result += left[i:]
  result += right[j:]
  return result

# Ranges to be used in the random array
n = [5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000]

# Run Merge Sort and collect the running time on the program
idx = 0
while (idx < len(n)):
  start = time.time()
  mergesort([random.random() for _ in range(n[idx])])
  end = time.time()
  runtime = (end - start)
  print("N: " + str(idx + 1), "Time: " + str(runtime))
  idx += 1
```

Insert Sort run time modification:

```
import time
import random

def insertsort(array):
    i = 0
    length = len(array)
    while(i < length):
        temp = array[i]
        j = i
        while(j > 0 and temp < array[j - 1]):
            array[j] = array[j - 1]
            j -= 1
        array[j] = temp
        i += 1
    return array

# Ranges to be used in the random array
n = [5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000]

# Run Merge Sort and collect the running time on the program
idx = 0
while (idx < len(n)):
    start = time.time()
    insertsort([random.random() for _ in range(n[idx])])
    end = time.time()
    runtime = (end - start)
    print("N: " + str(idx + 1), "Time: " + str(runtime))
    idx += 1
```

4b - **Run times on the FLIP school server**

mergeTime.py

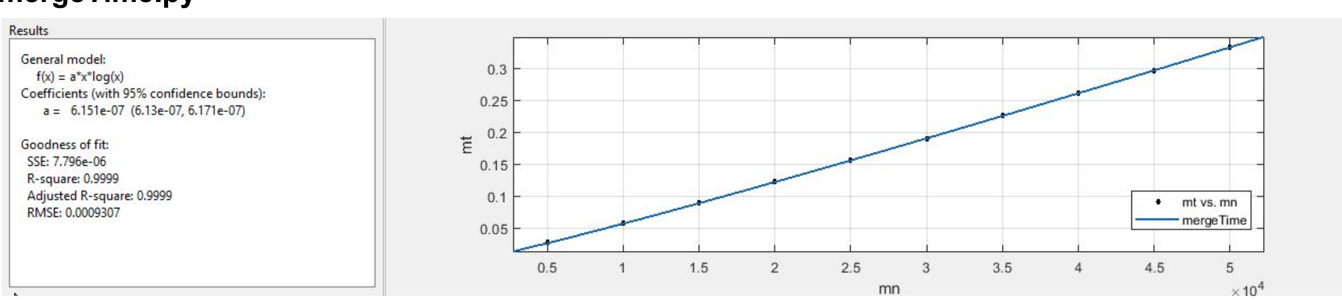
N	T
5000	0.0271799564362
10000	0.0576601028442
15000	0.0900650024414
20000	0.123214006424

25000	0.155815124512
30000	0.18927192688
35000	0.225196838379
40000	0.260485172272
45000	0.295611143112
50000	0.333196878433

insertTime.py

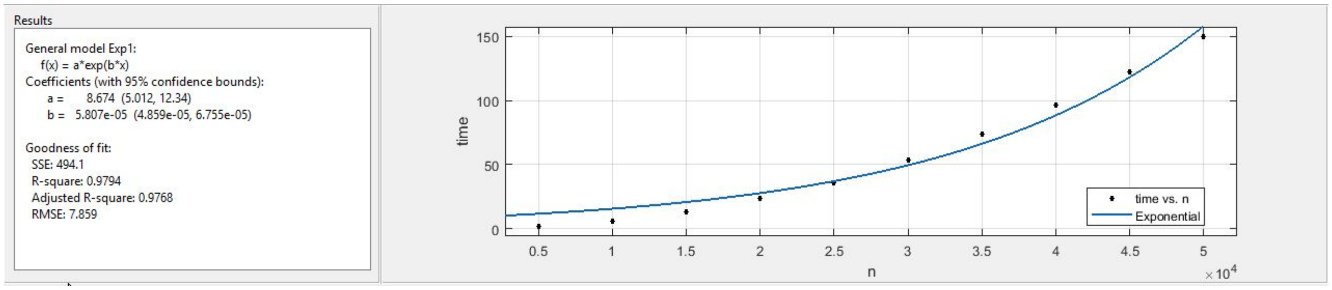
N	T
5000	1.47103905678
10000	5.85689496994
15000	12.7701561451
20000	23.6061120033
25000	36.1263029575
30000	53.7584118843
35000	74.1469941139
40000	96.1506781578
45000	121.999218941
50000	150.14786005

4c - **Plot data and fit a curve**
mergeTime.py



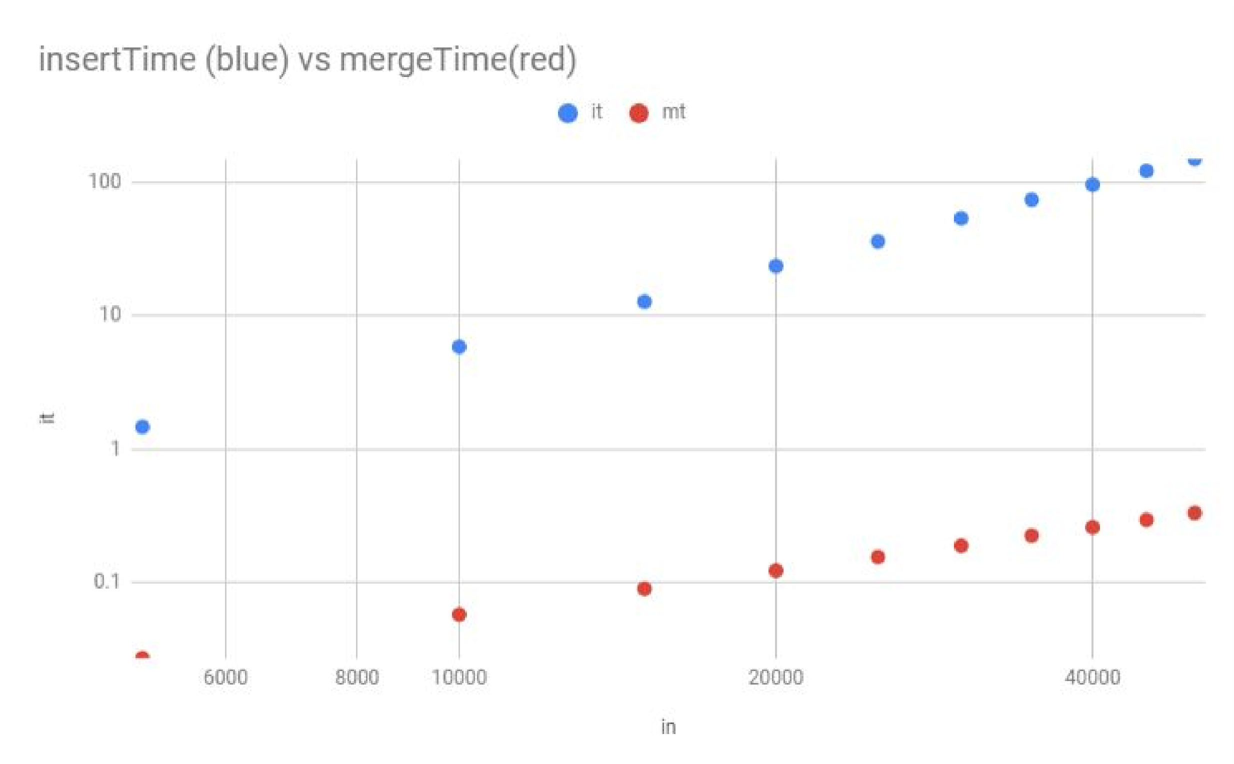
- A curve using n * log(n) best fits the data above (where mt = time and mn = size).

insertTime.py



- An exponential curve best fits the data above. f(x) = a * (b^n)

4d - **Combine data plots**



4e - **Comparison**

Comparing the two curves of merge and insert sort to their average cases (logn*n & n^2, respectively), I found there weren't any anomalies. The curve for merge sort using a custom equation of n*logn created a curve that fit the run time data perfectly. The same is true of insert sort which used an exponential curve (equation being c * b^n where C is some coefficient).