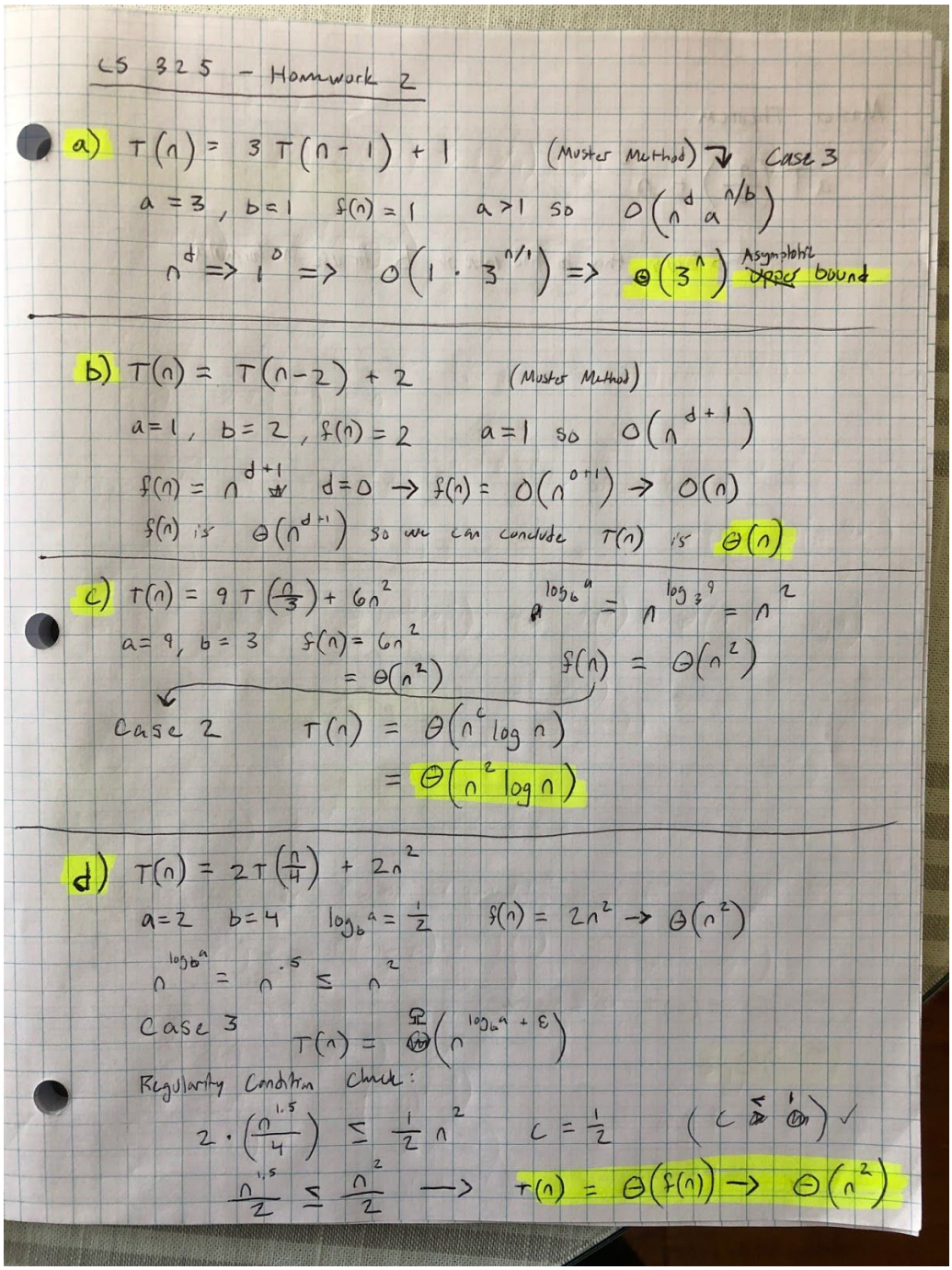
Christopher Matian
CS325 - Spring 2019
04/14/2019

Problem 1:



Problem 2:

A) A ternary search is essentially a binary search but instead of splitting into 2 search groups, it uses 3 search groups (divides the array into 3 instead of 2). The pseudo-code would look like the following:

```
tenary(array, search)
      If (length < 1)
            return error
      If (length == 1)
            return array[0] == search ? true : false
      If (length == 2)
            For in loop
                  If array[i] == search
                        return true
                  else
                        return false
      T1 = n/3
      T2 = 2n/3
      if(array[0] >= search AND search < array[T1]
            return ternary(array[0:T1, search]
      if (array[T1] <= search AND search < array[T2]
            return ternary(array[T1:T2], search)
      if (array[T2] <= search AND search < A[n-1]
            return ternary(array[T2: n], search)
      return false
```

2b) The recurrence of ternary search is

$T(n) = T\left(\frac{2n}{3}\right) + c(1)$ ← 1 operation for division

Worst case we make $\frac{2n}{3}$ recursive calls.

Best case we make $\frac{n}{3}$ recursive calls

2c) $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1 \quad b = \frac{3}{2} \quad c = 0$

$aT\left(\frac{n}{b}\right) + n^c$

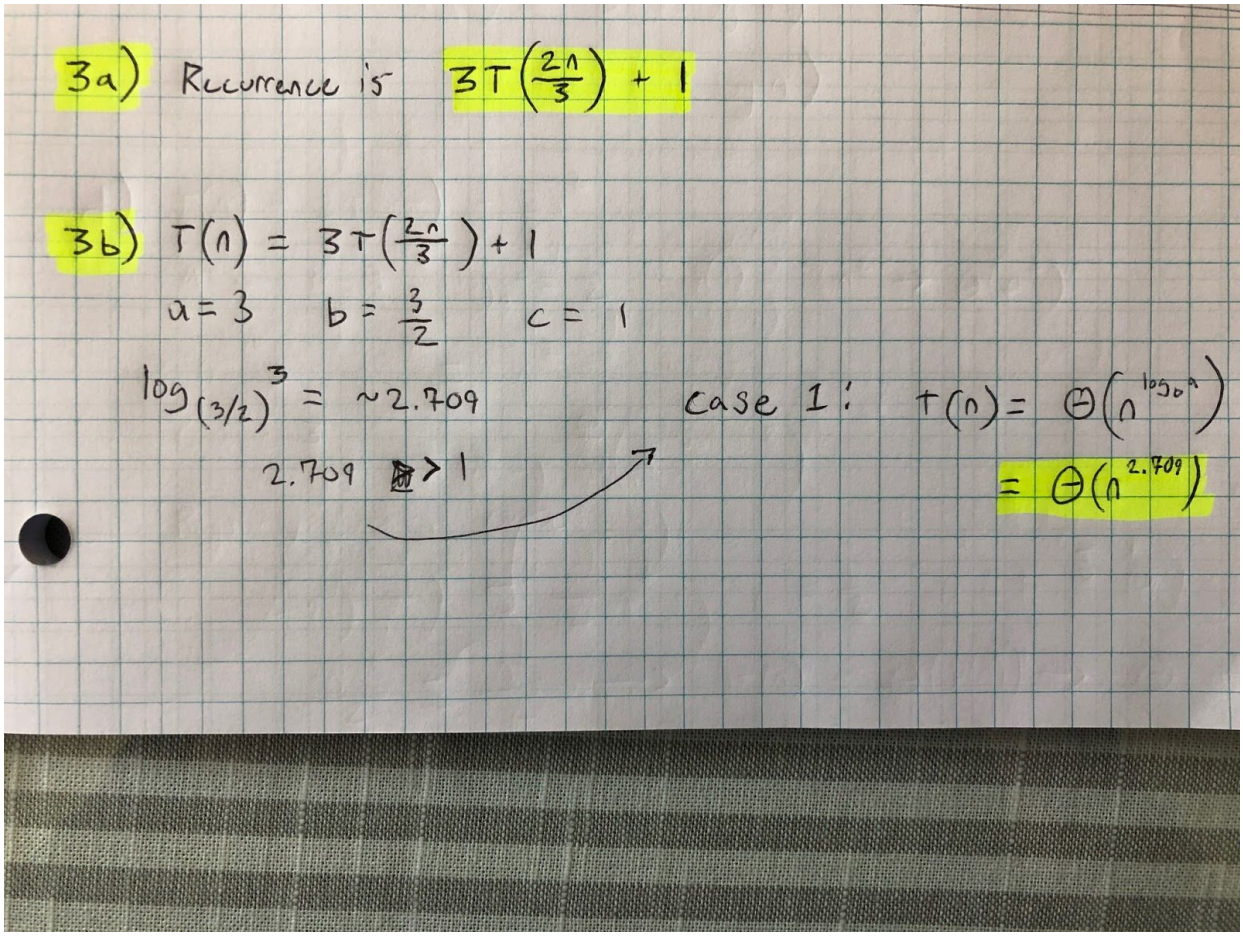
$\log_b{}^a = \log_{(3/2)}{}^1 = 0 = c$

Case 2:

$T(n) = \Theta(n^0 \log n)$

$= \Theta(\log n)$

Problem 3:


3a) Recurrence is $3T\left(\frac{2n}{3}\right) + 1$
3b) $T(n) = 3T\left(\frac{2n}{3}\right) + 1$
$a = 3 \quad b = \frac{3}{2} \quad c = 1$
$\log_{(3/2)} 3 = \sim 2.709$
$2.709 > 1$
case 1: $T(n) = \Theta\left(n^{\log_b a}\right)$
$= \Theta\left(n^{2.709}\right)$

Problem 4:

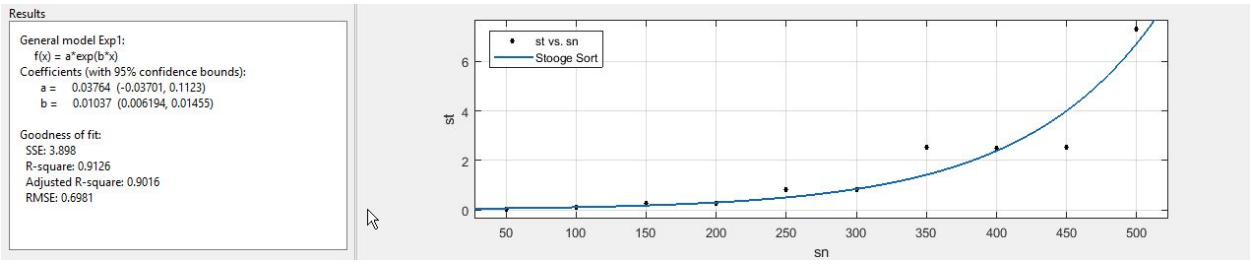
C) Run times on the FLIP school server

stoogeTime.py

N	T
50	0.0106010437
100	0.09317207336
150	0.2707030773
200	0.2714161873
250	0.8372240067
300	0.8203611374
350	2.522836208
400	2.479743004
450	2.523155928
500	7.320059061

D) Plot data and fit curve

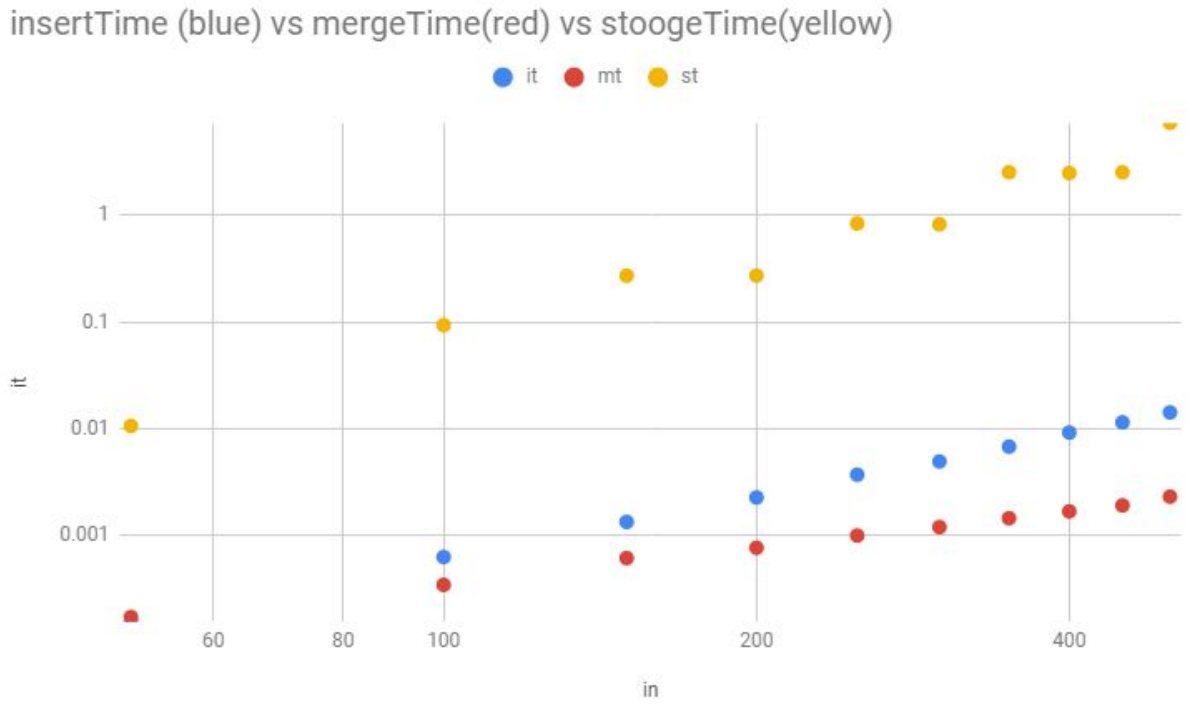
Zoom in to view the details



E) Comparison

Stoogesort's experimental running time complexity is about n^2.709 and the graph above is fit using a exponential curve (a*exp(b^n). There are no anomalies in comparison between the experimental running time and the tightest fitting equation/curve.

F) Combine


insertTime (blue) vs mergeTime(red) vs stoogeTime(yellow)
it
mt
st
1
0.1
0.01
0.001
it
60
80
100
200
400
in