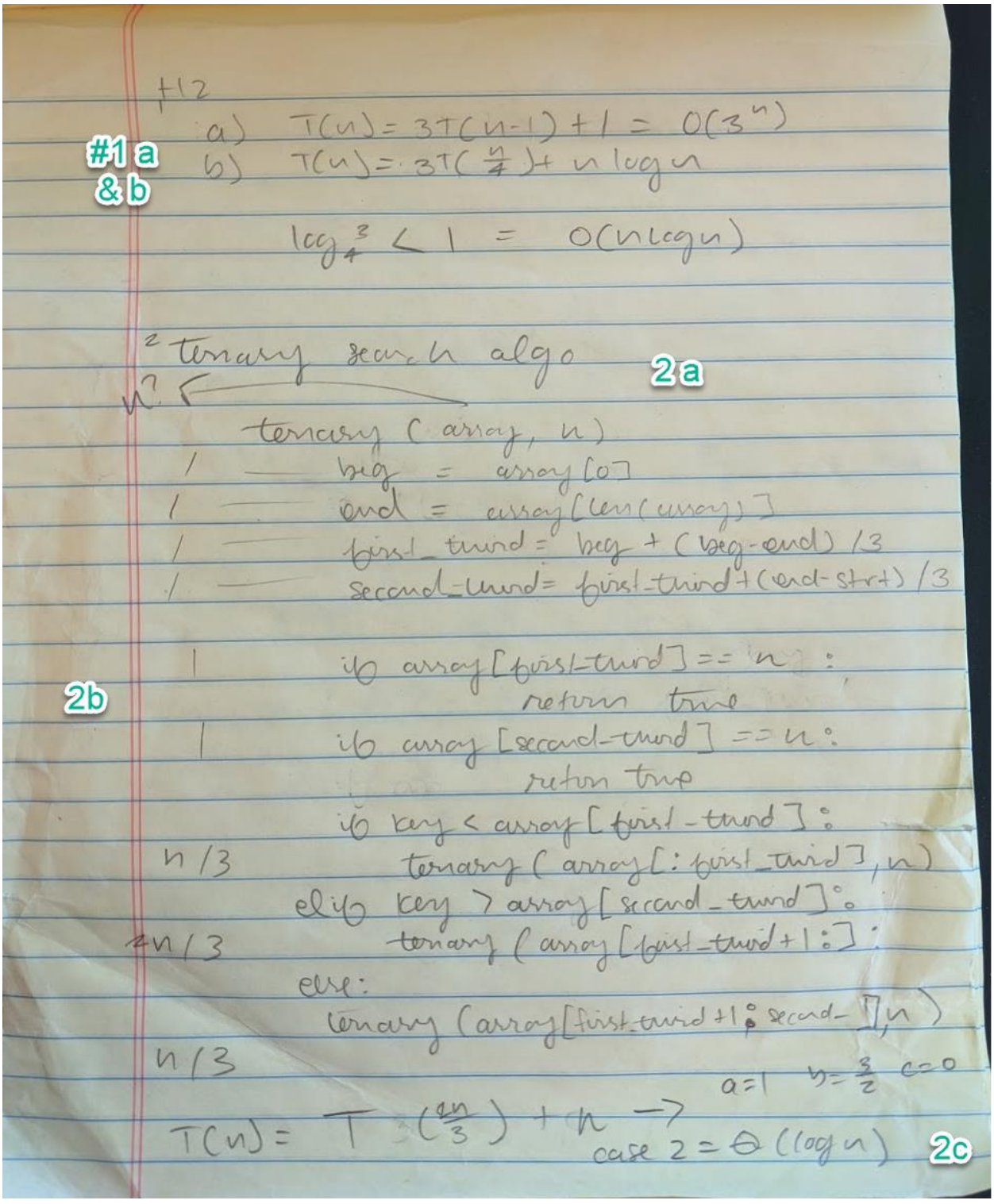
Bryan Rodriguez

CS 325 Fall 2020

HW 2

1



2a) a ternary search is much like a binary search; however, it splits the array into 3 groups.

The pseudo code would look something like this:

- Consider edge cases, an empty list and list with one index, a list with 2 indexes and return true or false accordingly.
- If the object you're searching for is greater than the object at the beginning of the list and less than the object at the end of the first third of the array, recurse into the first third of the array.
- If the object you're searching for is greater than the index at the end of the first third of the array, and less than the index at the second (2n/3) third of the array, recurse into the second third of the array
- Otherwise recurse into the last third of the array

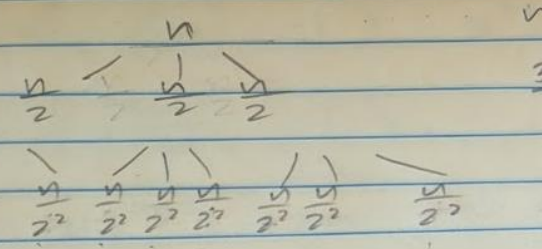
2b) the recurrence of ternary is $T(n) = T(2n/3) + n$, the worst case is 2n/3 and best is

2c) its runtime is $O(\log n)$

3a

3. $T(n) = 3T(\frac{n}{2}) + n$

a)



assume $\frac{n}{2^k} = 1$ $\quad \frac{n}{2^k}$ $\quad \frac{3^k n}{2^k} = (\frac{3}{2})^k n$

$n = 2^k$

$k = \log n$

$T(n) = (\frac{3}{2})^{\log n} \cdots \frac{3^4 n}{2^4} + \frac{3^3 n}{2^3} + \frac{3^2 n}{2^2} + \frac{3n}{2} + n$

$T(n) = n\left[\frac{3}{2}^{\log n} \cdots \frac{3^4}{2^4} + \frac{3^3}{2^3} + \frac{3^2}{2^2} + \frac{3}{2} + n\right]$

$T(n) = n \cdot \left(\frac{1 \cdot \frac{3}{2}^{\log n + 1} - 1}{\frac{3}{2} - 1}\right)$

$T(n) = 2n\left(\frac{\frac{3}{2} \cdot \frac{3}{2}^{\log n} - 1}{\frac{1}{2}}\right)$

$T(n) = 2n\left(\frac{3}{2} \cdot \frac{3^{\log n}}{n} - 1\right) \quad n = 2^k$

$= 2n\left(\frac{3}{2} \cdot \frac{n^{\log 3}}{n} - 1\right) \quad 3^{\log n} = n^{\log 3}$

$= 2n\left(\frac{3n^{\log 3}}{2n} - 1\right)$

$T(n) = 3n^{\log 3} - 2n \Rightarrow O(n^{\log 3})$

inductive Proof

$T(n) \leq n^{\log_2 3}$

3b

base $T(1) = 1$

$T(1) \leq 1^{\log_2 3}$ ✓

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

inductive $T(n) \leq 3 \cdot T\left(n^{\log_2 3} / 2^{\log_2 3}\right) + n$

$T(n) \leq 3\, T\left(n^{\log_2 3} / 3\right) + n$

$T(n) \leq T n^{\log_2 3} + n \leq C n^{\log_2 3}, \quad n \geq 1$

(D) $T(n) = O(n^{\log_2 3})$

4 a)

the recurrence for strassen's:

$$3T\left(\frac{3n}{2}\right) + (1)$$ 4a

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$f(n) = \Theta(n^k \log^p n)$

$\therefore a = 3$

$b = 2/3$

$f(n) = 1$

$n \log_b a = n \log_{2/3} 3 =$

$n^{2.709511} = O(n^{2.709511})$

4b

5b)

```
import time
import random

# Stoogesort code base pulled from geeks for geeks https://www.geeksforgeeks.org/
stooge-sort/


def stoogesort(arr, l, h):

    if l >= h:
        return

    # If first element is smaller
    # than last, swap them
    if arr[l] > arr[h]:
        t = arr[l]
        arr[l] = arr[h]
        arr[h] = t

    # If there are more than 2 elements in
    # the array
    if h-l + 1 > 2:
        t = (int)((h-l + 1)/3)

        # Recursively sort first 2 / 3 elements
        stoogesort(arr, l, (h-t))

        # Recursively sort last 2 / 3 elements
        stoogesort(arr, l + t, (h))

        # Recursively sort first 2 / 3 elements
        # again to confirm
        stoogesort(arr, l, (h-t))


# Ranges to be used in the random array
n = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]

# Run Stooge Sort and collect the running time on the program
counter = 0
while counter < len(n):

    arr = [random.randint(0, 10000) for i in range(n[counter])]
    size = len(arr) - 1
```

```
    start = time.time()
    stoogesort(arr, 0, size)
    stop = time.time()
    print("Sort size " + str(n[counter]), "Run time: " + str(stop - start))
    counter += 1
```

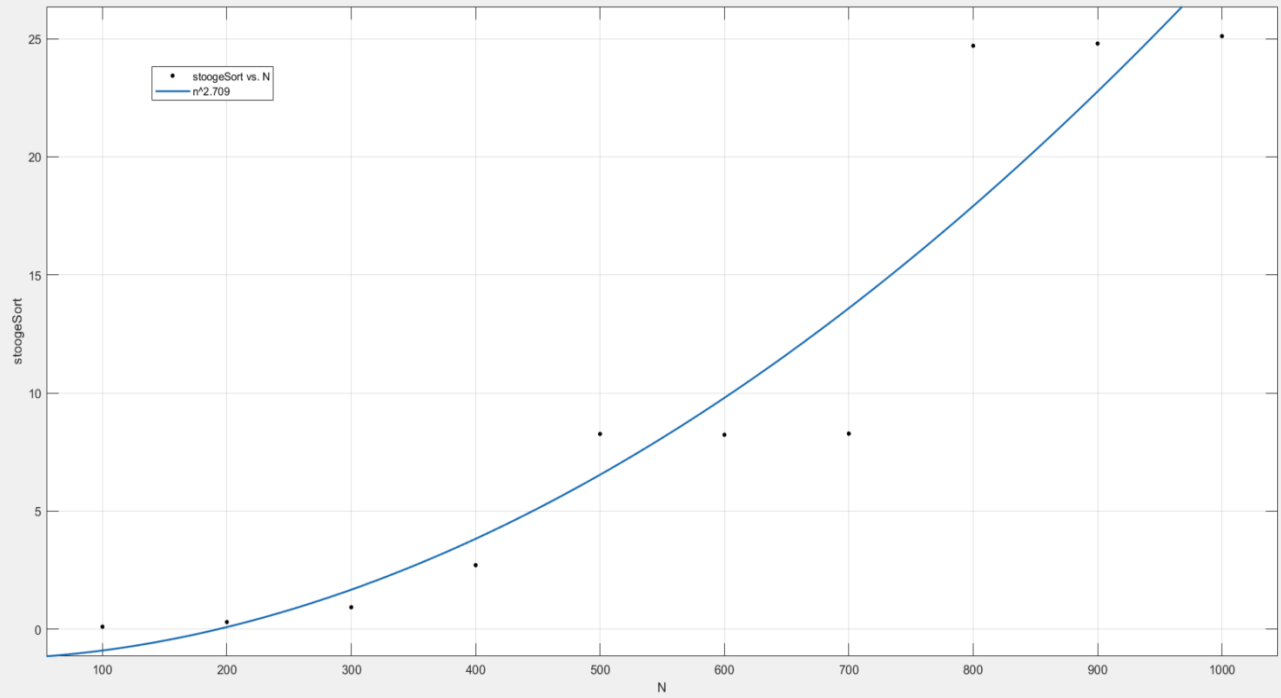
5c)

N	StoogeSort(nlog_2^3)
100	0.1031994
200	0.301194
300	0.9277961
400	2.7087297
500	8.2646515
600	8.2296929
700	8.27911329
800	24.70238733
900	24.79684711
1000	25.11054397

Results

General model Power2:
f(x) = a*x^b+c
Coefficients (with 95% confidence bounds):
a = 5.317e-05 (-0.0005002, 0.0006065)
b = 1.914 (0.4168, 3.412)
c = -1.263 (-8.54, 6.015)

Goodness of fit:
SSE: 95.83
R-square: 0.9041
Adjusted R-square: 0.8766
RMSE: 3.7



The type of curve that best fits the StoogeSort curve is an exponential one. From Matlab we can see that the formula is (a*x^b+c) or a polynomial of degree 2, compared to the theoretical run time of StoogeSort of n^2.709 this is comes close to the theoretical. I feel that I may be able to get a closer bound if I tweak the way I collect the run time of the sort within my code and potentially include larger array sets.

The graph including the merge and insert sort would likely read a little better if I ran the sorts against the same values, however, the other two sorts were too fast to get correct readings. I therefore plotted StoogeSort's time on the right axis and left merge and insert sort's time on the left axis.

stoogeSort Readme

1. Open the stoogeSort.py file and make sure your data.txt file is in the proper directory for it to be imported.
2. The stoogeSort.py file will output a stooge.out.txt file with the sorted arrays from data.txt
3. The first number in the arrays passed to the sort is assumed to be the total number of elements in the array and will thus be omitted.