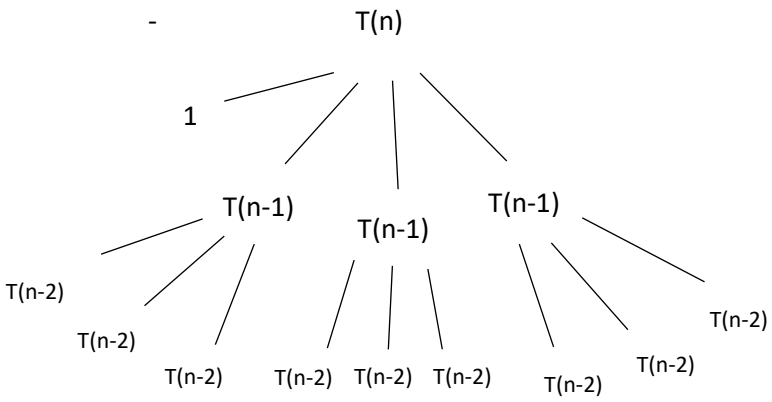
Casey Levy – CS 325 – HW 2

Problem 1

a) $T(n) = 3T(n-1) + 1$

- $T(n) = \begin{cases} 1 & n = 0 \\ 3T(n-1) + 1 & n > 0 \end{cases}$

- 

Level Cost

1

3^1

3^2

We continue the level cost until we reach $\mathbf{3^k}$.

$1 + 3^1 + 3^2 + 3^3 \ldots\ldots + 3^k \quad = \quad 3^{k+1} - 1$

Assume **n – k = 0**

∴ n = k

$= 3^{n+1} - 1$

$= \mathbf{O(3^n)}$

b) $T(n) = 3T(n/4) + n \log n$

- Using the Master Theorem we see a = 3, b = 4, $f(n) = n \log n$
- One case of the multiple cases says that if $f(n) = \Omega(n^c)$ for $c > \log_b a$ then $T(n) = \theta(f(n))$
- Using that case, $f(n) = \Omega(n^1)$, where $1 > \log_4 3$, therefore $f(n) = \theta(n \log n)$
- **O(n log n)**

Problem 2

a) 1. We define the function with four parameters as input, **l** and **r** for the "left" and "right" portions of the array, **x** for the element we want to find (key), and **array** for the sorted array we need to work on.
2. We check to see if **r >= 1**, if so, we continue. If not, program returns -1 and terminates

3. We create **mid_1** and **mid_2** to break the input array into three parts
4. Check to see if the key is in either of the two mids we just created. If found in either, we return the desired value and the program ends successfully. If not, we continue.
5. If value is not found in step above, we perform conditionals to determine where the key is located and then recursively call **ternarySearch** within that section of the array
6. Value is found and returned and the program terminates successfully.

```
1.  def ternarySearch(l, r, x, array):
2.      if (r >= 1):
3.          mid_1 = l + (r – 1) //3
            mid_2 = r + (r – 1) //3

4.          if (array[mid_1] == x):
                return mid_1

            if(array[mid_2] == x):

                return mid_2

5.          if (x < array[mid_1]):
                return ternarySearch(l, mid_1 – 1, x, array)

            elif (x > array[mid_2]):
                return ternarySearch(mid_2 + 2, r, x, array)

            else:
                return ternarySearch(mid_1 + 1, mid_2 – 1, x, array)

        return -1
```

b) T(n) = T(n/3) + O(1)

c) At each step of the function, we are reducing the range of search by 3. after **n** steps, if we are to find our desired element, the size would be $\mathbf{3^n}$. This equates to the runtime being $\mathbf{O(\log_3 n)}$.

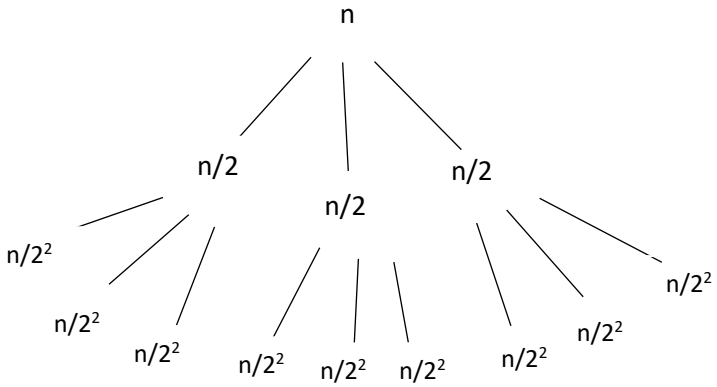
Asymptotic Runtime: $\mathbf{O(\log_3 n)}$

Since binary search breaks the search range into two halves and ternary breaks it into three portions, the worst case runtime for a binary search is $\mathbf{O(\log_2 n)}$ vs the ternary search runtime being $\mathbf{O(\log_3 n)}$. Although ternary search performs less iterations, this does not make up for the fact that it also must perform more comparisons compared to binary search which performs less comparisons. This makes **binary search** the preferred search method where applicable.

Problem 3

a) Tree Method for **T(n) = 3 · T(n/2) + n**

$$T(n) = \begin{cases} 1 & n = 1 \\ 3 \cdot T(n/2) + n & n > 1 \end{cases}$$



Level Cost

n

n

n

We continue the tree and level cost until we reach $\mathbf{n/2^k}$

The level cost is **n** every time the function runs

Assume $\mathbf{n/2^k = 1}$

$\mathbf{\therefore\ n = 2^k}$

k = log n

= **O(n log n)**

b) Master Method for **T(n) = 3 · T(n/2) + n**

- Master Method – T(n) = aT(n/b) + f(n)
- (Taken from Abdul Bari's YouTube video titled "2.4.1 Masters Theorem in Algorithms for Dividing Function #1") we see that in the Master Method, when a ≥ 1 and b > 1, then f(n) = $\theta(n^k \log^p n)$
- **Case #1: if $\log_b a > k$, then $\theta(n^{\log_b a})$**
- $\log_b a = \log_2 3 = 1.58$
- k = 1
- p = 0
- 1.58 > 1, therefore case #1 is satisfied where runtime is $\theta(n^{\log_b a})$
- $\theta(n^{\log_b a}) = \theta(n^{\log_2 3}) = \theta(n^{1.58})$
- **Case #1 is satisfied with a runtime of $\theta(n^{1.58})$**

Problem 4

a) Number of Comparisons

- n = 1, comparisons = **0**
- n = 2, comparisons = **1**
- n = 3, comparisons = 3 * 1 + 0 = **3**
- n = 4, comparisons = 4 * 2 + 1 = **9**
- n = 5, comparisons = 5 * 3 + 2 = **17**
- this gives us **n * (n-2) + (n-3)**
- **Recurrence is T(n) = 3T(3n/2) + θ(1)**

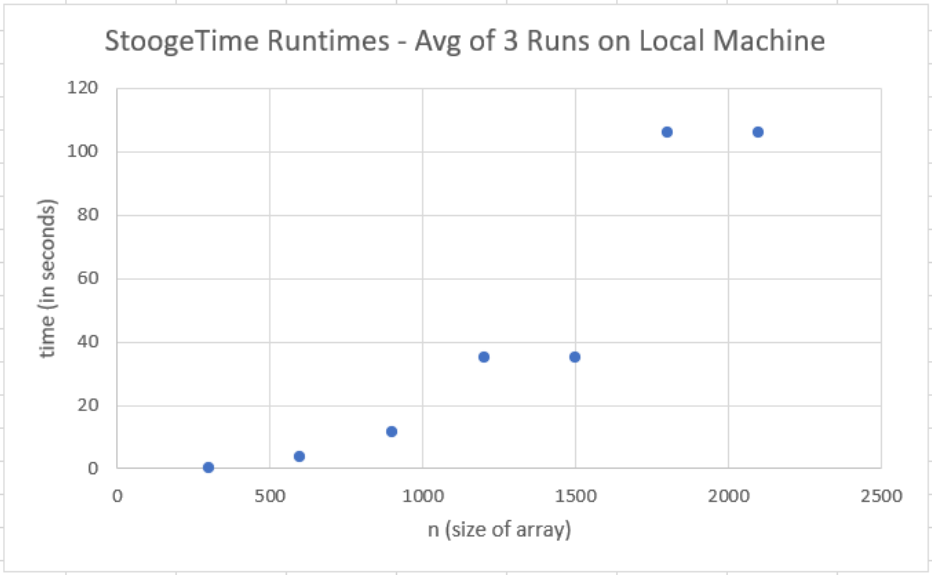
b) Solving T(n) = 3T(3n/2) + θ(1)

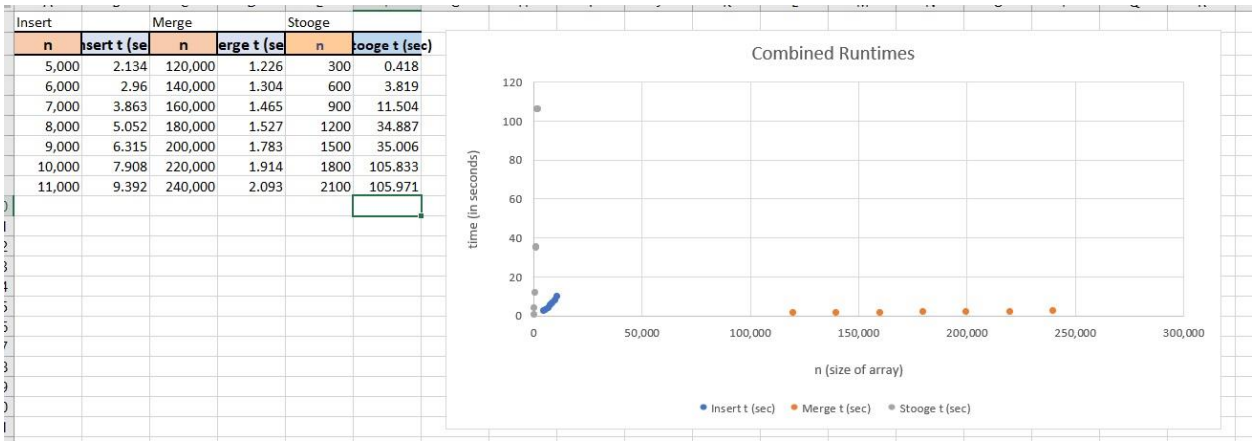
- We can use Master Method here - T(n) = aT(n/b) + f(n)
- when a ≥ 1 and b > 1, then f(n) = $\theta(n^k \log^p n)$
- Case #1: if $\log_b a > k$, then $\theta(n^{\log_b a})$
- $\log_b a = \log_{2/3} 3 = 2.709$
- **Runtime is $O(n^{2.709})$**

Problem 5

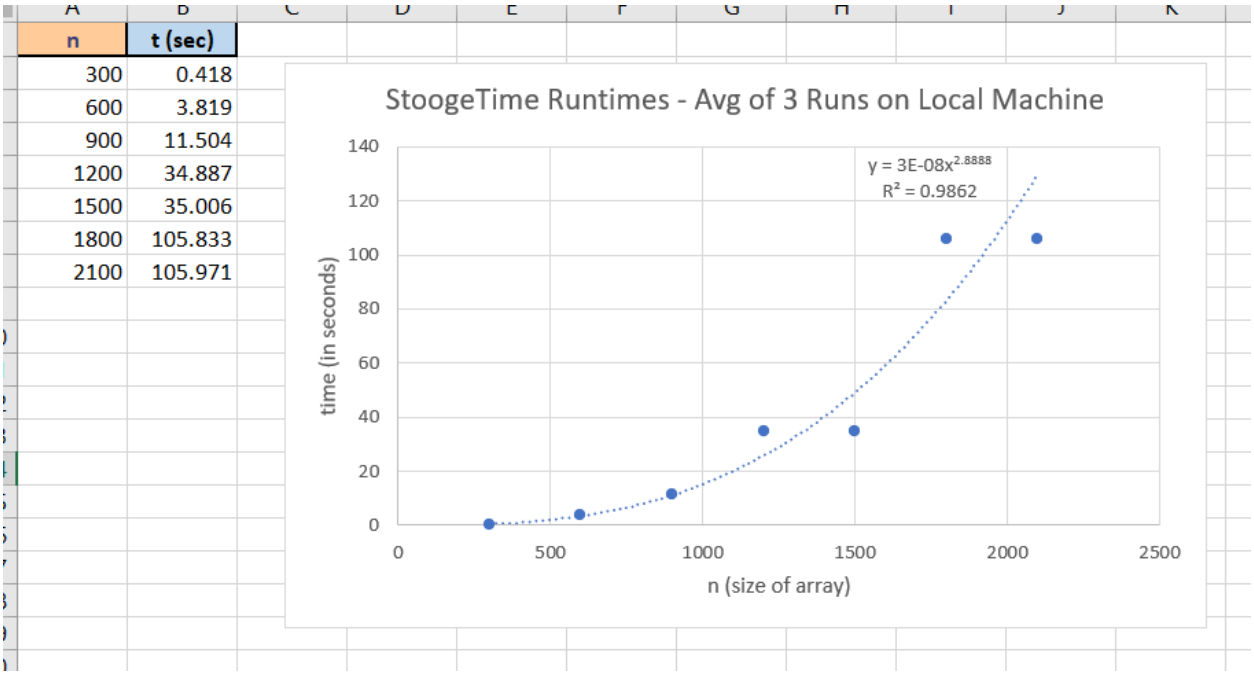
c)

n	t (sec)
300	0.418
600	3.819
900	11.504
1200	34.887
1500	35.006
1800	105.833
2100	105.971





d)



- With the theoretical runtime being $n^{2.709}$, based on my actual runtimes, my data was very similar to the theoretical runtime of this program. I had noticed once I plotted my data on the combined graph with insert and merge sort programs, how similar the trendline was for stooge sort, compared to the runtime of a function with n^2 runtime.