Pranav Prabhu

Problem 1

a) Pseudocode for merge_sort3:

```
merge(array, begin, m1, m2, end):
    left_side_array = array[begin : m1]     # begin to midddle1
    middle_array = array[middle1 : middle2 - 1]
    right_side_array = array[middle2 - 1 : end]

    left_index = middle_index = right_index = 0

    for element in entire_array:
        left_arr_index = left_arr[left_index]
        middle_arr_index = middle_arr[middle_index]
        right_arr_index = right_arr[right_index]

        minimum_value = min([left_arr_index,
   middle_arr_index, right_arr_index])      # find the minimum

        if (minimum_value == left_arr_index):
            array[i] = left_arr_index
            left_index += 1
        elif (minimum_value == middle_arr_index):
            array[i] = middle_arr_index
            middle_index += 1
        else:
            array[i] = right_arr_index
            right_index += 1

merge_sort(array, begin, end):
    if (end - begin < 2):
        return array
    else:
        middle1 = begin + ((end - begin) // 3)
        middle2 = begin + 2 * ((end - begin) // 3) + 1

        merge_sort(array, begin, middle1)
        merge_sort(array, middle1, middle2)
        merge_sort(array, middle2, end)
        merge(array, begin, middle1, middle2 + 1, end)
        return array

main():
    open and read data_file
    array = []
    store integers in array
    grab correct n values from array
    merge_sort(num_arr, 0, len(array))
    for num in array:   print(num)
```

Pseudocode for merge3:

```
merge(array, begin, m1, m2, end):
    left_side_array = array[begin : m1]     # begin to midddle1
    middle_array = array[middle1 : middle2 - 1]
    right_side_array = array[middle2 - 1 : end]

    left_index = middle_index = right_index = 0

    for element in entire_array:
        left_arr_index = left_arr[left_index]
        middle_arr_index = middle_arr[middle_index]
        right_arr_index = right_arr[right_index]
```

```
          minimum_value = min([left_arr_index,
    middle_arr_index, right_arr_index])      # find the minimum

          if (minimum_value == left_arr_index):
               array[i] = left_arr_index
               left_index += 1
          elif (minimum_value == middle_arr_index):
               array[i] = middle_arr_index
               middle_index += 1
          else:
               array[i] = right_arr_index
               right_index += 1

merge_sort(array, begin, end):
     if (end - begin < 2):
          return array
     else:
          middle1 = begin + ((end - begin) // 3)
          middle2 = begin + 2 * ((end - begin) // 3) + 1

          merge_sort(array, begin, middle1)
          merge_sort(array, middle1, middle2)
          merge_sort(array, middle2, end)
          merge(array, begin, middle1, middle2 + 1, end)
          return array

main():
     for n in range(5000, 50001, 5000):      # [5000, 50000] in steps of 5000
          array = []

          for i in range(0, n+1):
               add random integers in range [0, 10000]

          begin_time()
          merge_sort(array, 0, len(array))
          endtime()
          print(end_time - begin_time)
```

b) Recurrence Relation:

In the case of 2-way merge sort, the recurrence relation would be:

T(n) = {O(1) if n = 1; 2T(n/2) + O(n) if n > 1}

In the case of 3-way merge sort, the recurrence relation would be:

T(n) = {O(1) if n = 1; 3T(n/3) + O(n) if n > 1}

c) Runtime:

Since A = 3 and B = 3, A = B, so we use **n*log(n)** or more specifically **n*log$_3$(n)**.

Problem 2

a) Pseudocode for stooge_sort:

```
stooge_sort(array, start, end):
     if start >= end: return

     if array[start] > array[end]:            # swap elements
          temp = array[start]
          array[start] = array[end]
          array[end] = temp

     length = end - start + 1
     if length >= 3:
          middle = length / 3
          # recursive calls to sort the array
```

```
        stooge_sort(array, start, end - middle)
        stooge_sort(array, start + middle, end)
        stooge_sort(array, start, end - middle)

main():
    open and read data_file
    array = []
    store integers in array
    grab correct n values from array
    stooge_sort(num_arr, 0, len(array))
    for num in array:   print(num)
```

b) Recurrence Relation:

The recurrence relation for Stooge Sort is as follows:

$\mathbf{T(n) = 3T(3n/2) + O(1)}$

c) Runtime:

Since A = 3 and B = 3/2 or 1.5, A > B, so we use $\mathbf{n^{\log_{B}(A)}}$ or more specifically $\mathbf{n^{\log_{1.5}(3)}}$. So, this would be log(3) / log(1.5), which rounds to approximately $n^{2.7095}$.

Problem 5

A. Table of Values:

Merge Sort Runtime:

n	time (sec)
5000	0.089254
10000	0.175154
15000	0.283495
20000	0.206053
25000	0.248575

30000	0.353791
35000	0.54033
40000	0.416439
45000	0.481791
50000	0.547842

Stooge Sort Runtime:

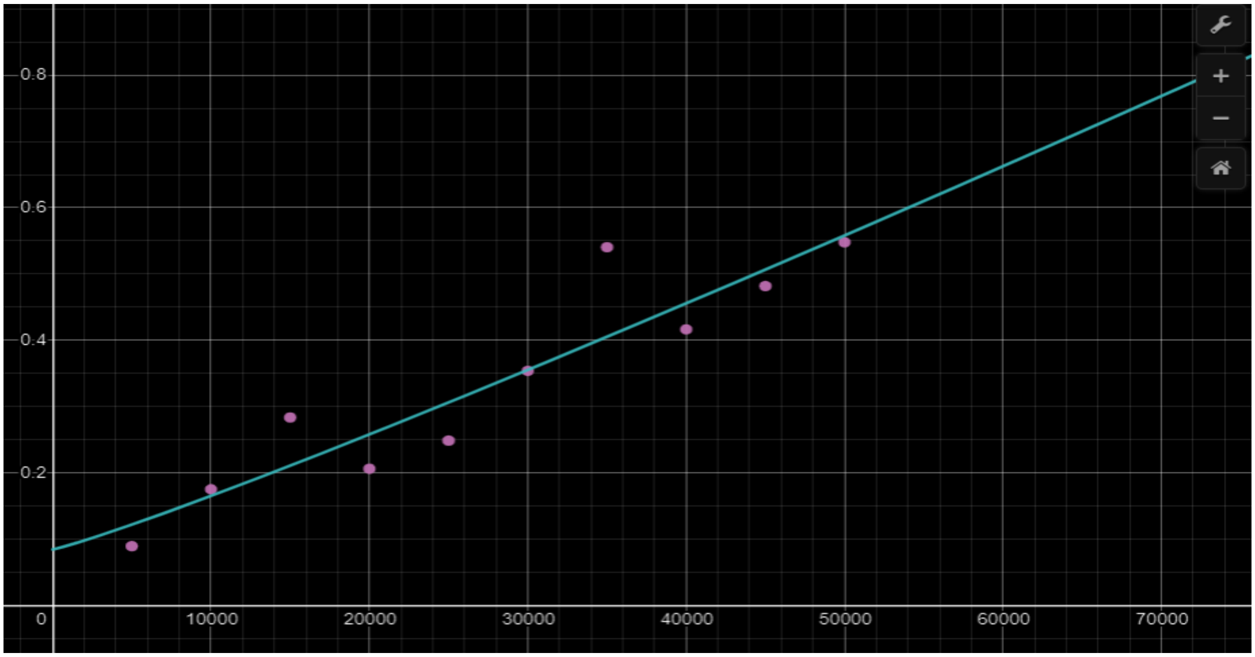
n	time(sec)
50	0.027717
100	0.247368
150	0.554662
200	0.388771
250	1.172593
300	1.175426
350	3.549613
400	3.515138
450	3.554631
500	10.61368
550	20.21804
600	19.73284
650	18.28422

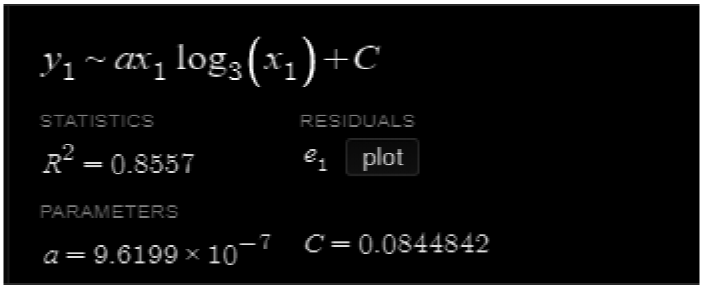
I took the average of multiple runs for each n value (array size). So, this could be the average case running time although average rarely happens. It could also be the worst case running time.

B. Graphs:

Merge Sort:

3 Way Merge Sort Running Time (desmos.com)





$y_1 \sim ax_1 \log_3(x_1) + C$

STATISTICS

$R^2 = 0.8557$

RESIDUALS

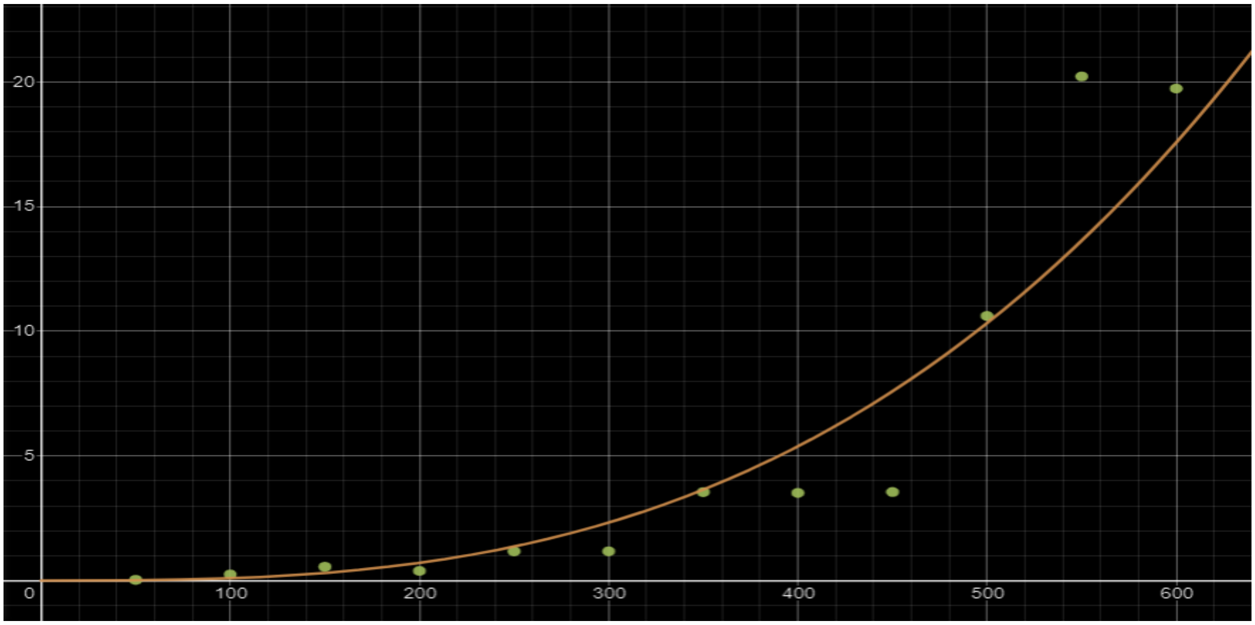
e_1

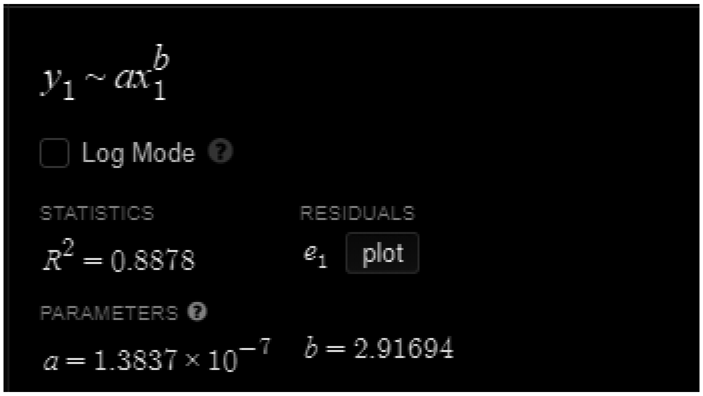
PARAMETERS

$a = 9.6199 \times 10^{-7}$

$C = 0.0844842$

Stooge Sort:





$y_1 \sim ax_1^b$

□ Log Mode

STATISTICS

$R^2 = 0.8878$

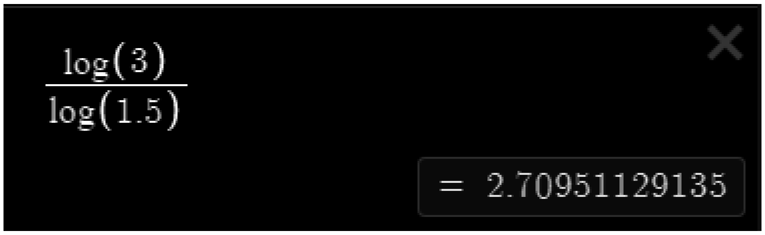
RESIDUALS

e_1 plot

PARAMETERS

$a = 1.3837 \times 10^{-7}$ $b = 2.91694$

Theoretical runtime for Stooge Sort:



$$\frac{\log(3)}{\log(1.5)} = 2.70951129135$$

Stooge Sort Runtime (desmos.com)