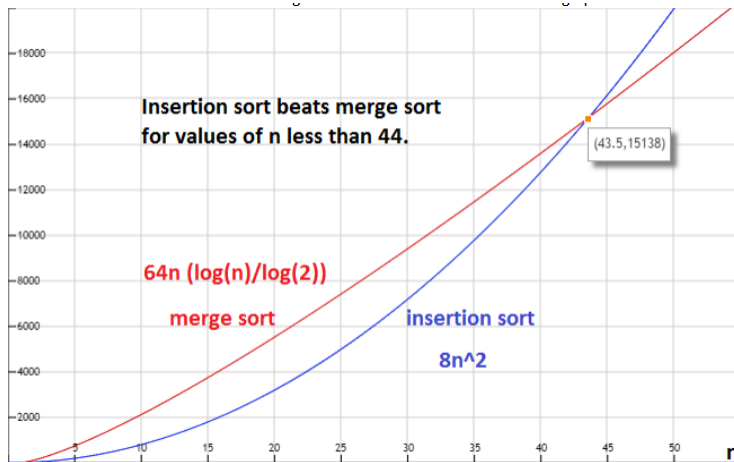


## CS 325 - HW 1 Solutions Examples

**Problem 1: 1 point - must include either a graph, table or some explanation as to how they got the result  $1 < n < 44$  insertion sort runs faster than merge sort ( $2 \leq n \leq 43$ )**



To solve this, I made a simple spreadsheet that quickly found that insertion sort above will beat for values of  $n < 44$ .

| value of n | $8n^2$  | $64n \lg n$ |
|------------|---------|-------------|
| 1          | 8       | 0           |
| 2          | 32      | 128         |
| 3          | 72      | 304         |
| 5          | 200     | 743         |
| 10         | 800     | 2126        |
| 20         | 3200    | 5532        |
| 40         | 12800   | 13624       |
| 43         | 14792   | 14933       |
| 44         | 15488   | 15374       |
| 45         | 16200   | 15817       |
| 50         | 20000   | 18060       |
| 100        | 80000   | 42521       |
| 1000       | 8000000 | 637810      |

## CS 325 - HW 1 Solutions Examples

### 2. 5 points - 0.5 point deduction for each one missed.

1) (5 pts) For each of the following pairs of functions, either  $f(n)$  is  $O(g(n))$ ,  $f(n)$  is  $\Omega(g(n))$ , or  $f(n) = \Theta(g(n))$ . Determine which relationship is correct and explain.

- |    |   |                    |                            |
|----|---|--------------------|----------------------------|
| a. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = n^{0.25};$ | $g(n) = n^{0.5}$           |
| b. | <b><math>f(n)</math> is <math>\Omega(g(n))</math></b> | $f(n) = n;$        | $g(n) = \log^2 n$          |
| c. | <b><math>f(n)</math> is <math>\Theta(g(n))</math></b> | $f(n) = \log n;$   | $g(n) = \ln n$             |
| d. | <b><math>f(n)</math> is <math>\Theta(g(n))</math></b> | $f(n) = 1000n^2;$  | $g(n) = 0.0002n^2 - 1000n$ |
| e. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = n \log n;$ | $g(n) = n\sqrt{n}$         |
| f. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = e^n;$      | $g(n) = 3^n$               |
| g. | <b><math>f(n)</math> is <math>\Theta(g(n))</math></b> | $f(n) = 2^n;$      | $g(n) = 2^{n+1}$           |
| h. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = 2^n;$      | $g(n) = 2^{2^n}$           |
| i. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = 2^n;$      | $g(n) = n!$                |
| j. | <b><math>f(n)</math> is <math>O(g(n))</math></b>      | $f(n) = \lg n;$    | $g(n) = \sqrt{n}$          |

## CS 325 - HW 1 Solutions Examples

3) a. **2 points: 1 for true (prove), 1 for proof**

If  $f_1(n) = \Theta(g(n))$  and  $f_2(n) = \Theta(g(n))$  then  $f_1(n) = \Theta(f_2(n))$ .

By definition if  $f_1(n) = \Theta(g(n))$  then  $c_1 g(n) \leq f_1(n) \leq c_2 g(n)$  for  $n \geq n_0$  (I)

By definition if  $f_2(n) = \Theta(g(n))$  then  $k_1 g(n) \leq f_2(n) \leq k_2 g(n)$  for  $n \geq n_2$  (II)

From (II) we obtain

$$g(n) \leq (1/k_1) f_2(n) \Rightarrow c_2 g(n) \leq (c_2/k_1) f_1(n) \quad (\text{III})$$

From (II) we can also obtain

$$(1/k_2) f_2(n) \leq g(n) \Rightarrow (c_1/k_2) f_1(n) \leq c_1 g(n) \quad (\text{IV})$$

Now by combining inequalities (I) (III) and (IV) we obtain

$$(c_1/k_2) f_2(n) \leq c_1 g(n) \leq f_1(n) \leq c_2 g(n) \leq (c_2/k_1) f_2(n) \quad (\text{V})$$

If we let  $c_3 = (c_1/k_2)$  and  $c_4 = (c_2/k_1)$  then (V) becomes

$$c_3 f_2(n) \leq f_1(n) \leq c_4 f_2(n) \quad \text{for } n_3 = \max\{n_0, n_2\}$$

Therefore by definition,  $f_1(n) = \Theta(f_2(n))$ .

b. **2 points: 1 for false (disprove), 1 for counterexample.**

If  $f_1(n) = O(g_1(n))$  and  $f_2(n) = O(g_2(n))$  then  $f_1(n) + f_2(n) = \Theta(g_1(n) + g_2(n))$

Many counterexamples:

If  $f_1(n) = n$ ,  $f_2(n) = n$ ,  $g_1(n) = n!$  &  $g_2(n) = n!$

## CS 325 - HW 1 Solutions Examples

### 4) 10 points total

README file – 1 point

Fully commented code - 1 points

Run code on TEACH with the file - Execution

4 points for the correct execution of insertion sort and

4 points for execution of merge sort.

data.txt containing the values below

10 10 9 8 7 6 5 4 3 2 1

3 1 1 1

5 9 8 2 3 3

merge.out and insert.out each should contain

1 2 3 4 5 6 7 8 9 10

1 1 1


2 3 3 8 9

### 5) 10 points total- Solutions may vary

a) 2 points - Insertion Sort and Merge Sort code with timing added (you do not have to run)

b) 2 points for data at least 5 values for each algorithm that are non-zero

Insertion Sort

| $x_2$ |  $y_2$ |
|-------|---|
| 0     | 0   |
| 10000 | 0.11  |
| 20000 | 0.45  |
| 30000 | 1.01  |
| 40000 | 1.82  |
| 50000 | 2.86  |
| 60000 | 4.1   |
| 70000 | 5.57  |
| 80000 | 7.24  |
| 90000 | 9.18  |

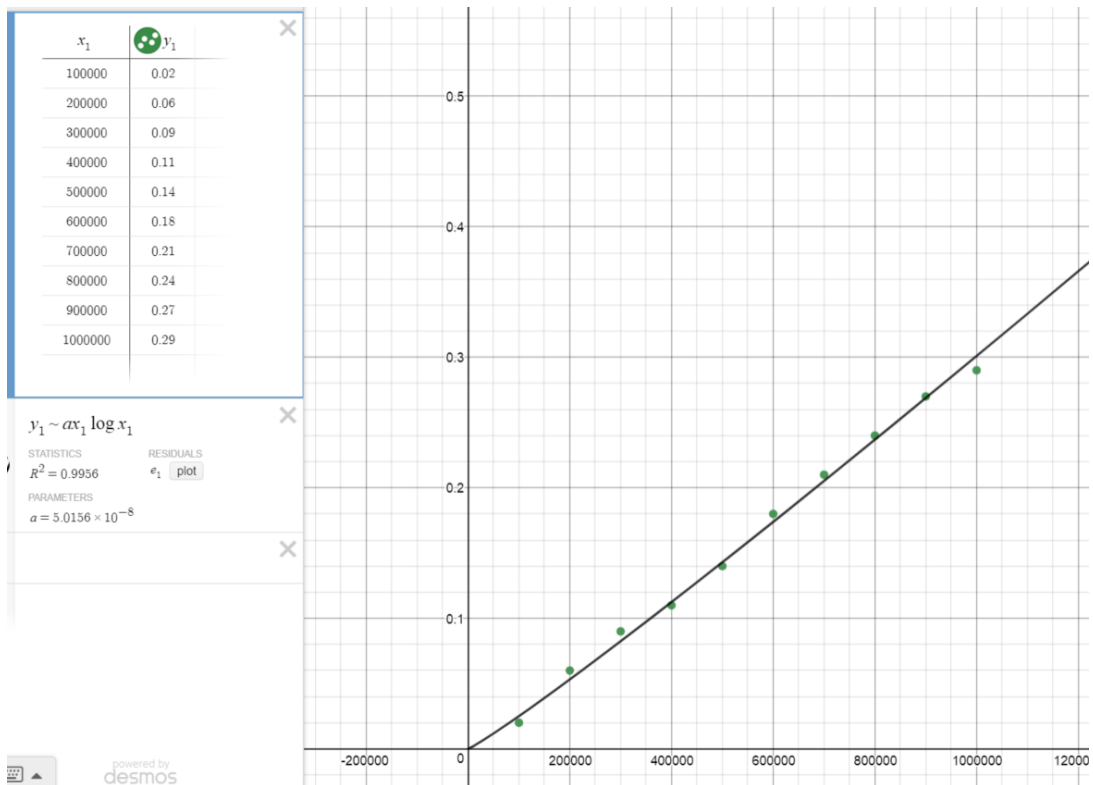
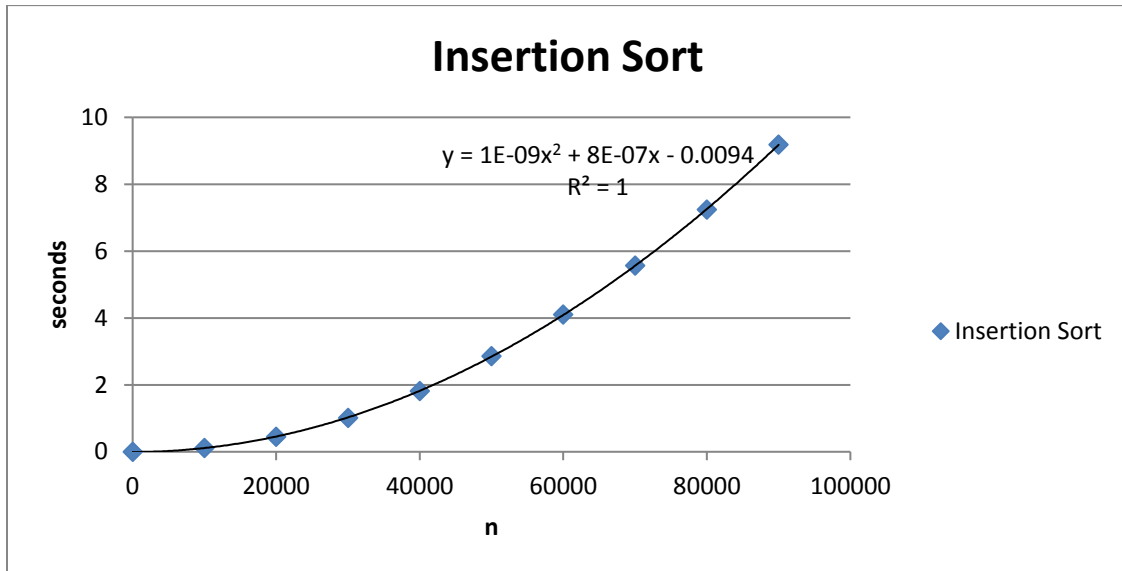
Merge Sort

| n       | Seconds |
|---------|---------|
| 0       | 0       |
| 100000  | 0.02    |
| 200000  | 0.06    |
| 300000  | 0.09    |
| 400000  | 0.11    |
| 500000  | 0.14    |
| 600000  | 0.18    |
| 700000  | 0.21    |
| 800000  | 0.24    |
| 900000  | 0.27    |
| 1000000 | 0.29    |

## CS 325 - HW 1 Solutions Examples

c) **4 points** - Plot data and fit a curve Use Excel Matlab, R or Desmos

- Insertion sort plot 1 pt
- Insertion sort curve 1 pt
- Merge sort plot 1 pt
- Merge sort curve 1 pt



Merge sort  $T(n) = (5.0156\text{E-}8)n \log n$  with  $R^2 = 0.9956$

## CS 325 - HW 1 Solutions Examples

**Insertion sort** has a quadratic fitted curve is displayed on the graph,

$$T(n) = 1E-09n^2 + 8E-07n - 0.0094$$

$$R^2 = 1$$

This is an almost perfect fit. (Results may vary)

**Merge sort** looks linear (or  $n \log n$ ) fitted line is on the graph. Full credit for fitting an  $n \log n$  curve

$$y = (5.0156E-8)n \log n$$

$$R^2 = 0.9956$$

This is also a very good fit.

**d) 1 point for combined graph**

**e) 1 point comparison**

**Insertion sort** – average experimental running time is  $\Theta(n^2)$  which matches the theoretical value

**Merge sort** – average experimental running time is  $\Theta(n)$  which differs from the theoretical value of  $\Theta(n \log n)$ . Answers may vary.

### Extra Credit

*Must include verbal explanation for full credit*

**+2 if all analysis and graphs are done for both algorithms with best case data.**

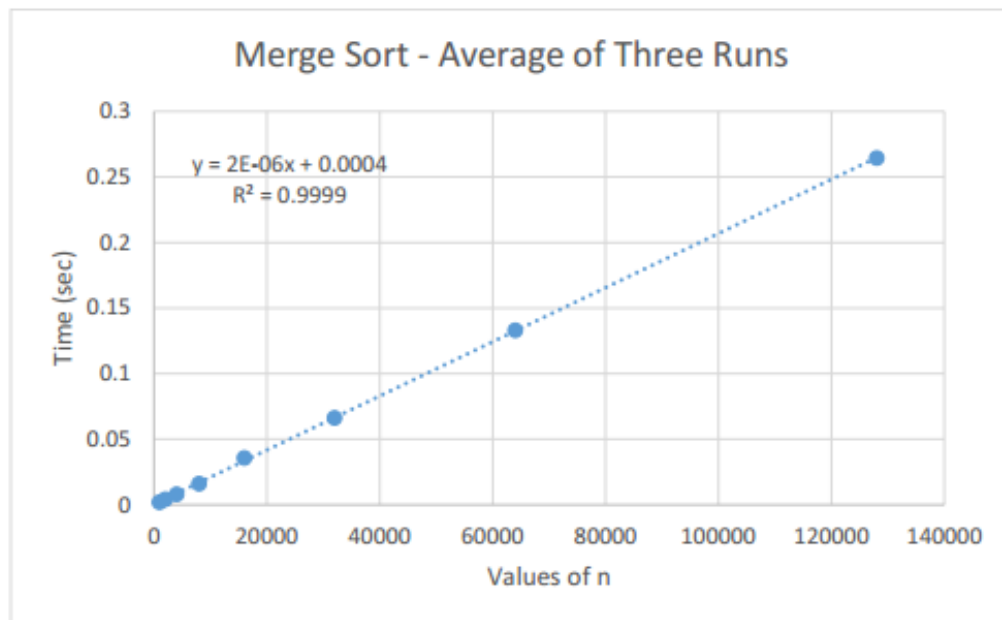
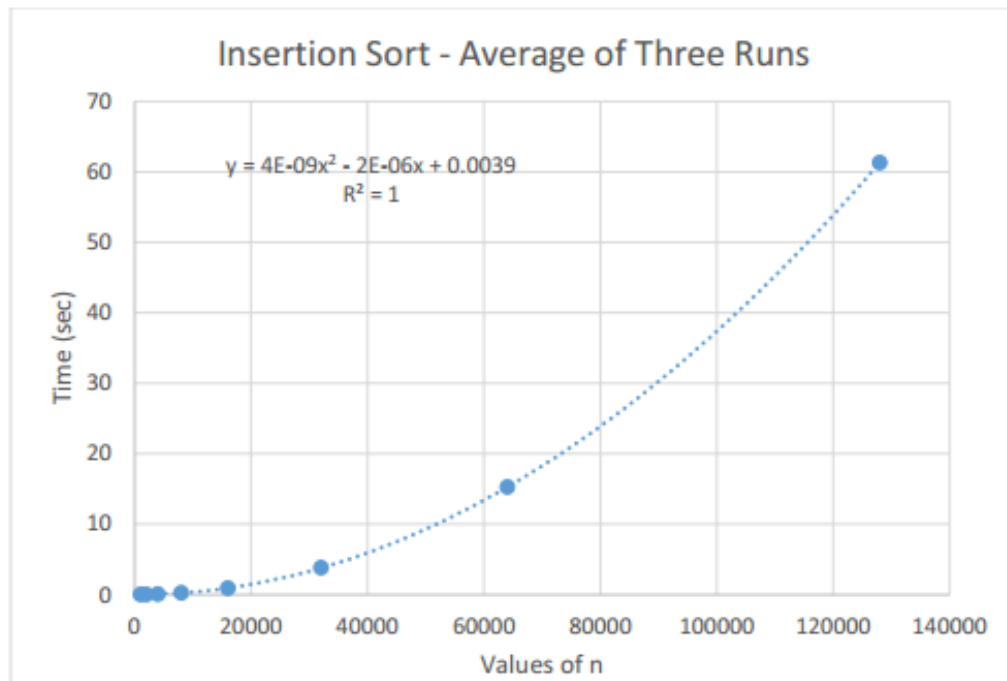
Best case the input array is already sorted. Insertion Sort will be linear and will run faster than Merge Sort which should have results similar to the average case.

**+2 if all analysis and graphs are done for both algorithms with worst case data.**

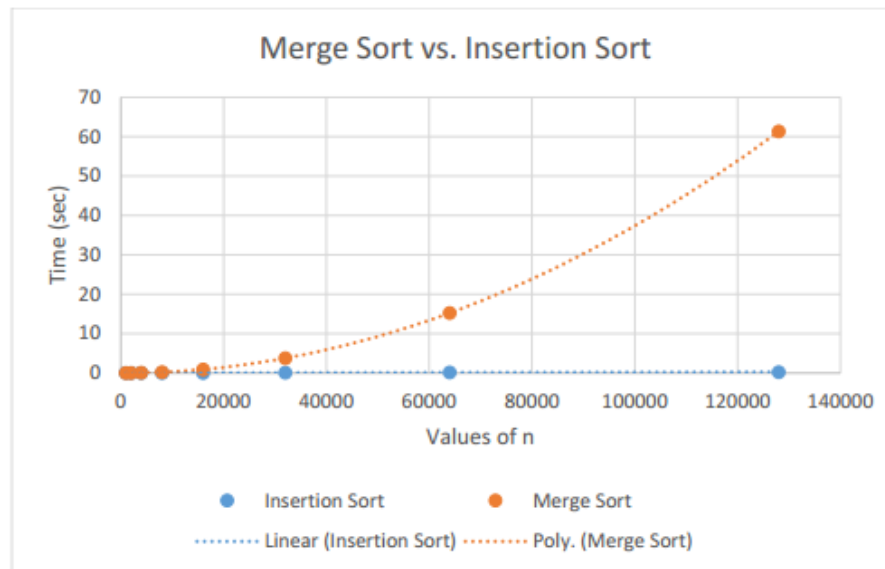
Worst case the input array in reverse order. Insertion Sort will be quadratic and will probably slower than in the average analysis. Merge Sort which should have results similar to the average case.

## CS 325 - HW 1 Solutions Examples

### Student example submissions



## CS 325 - HW 1 Solutions Examples



The graph that represents the data is best is the combination graph displaying the results of both sorting methods, as it clearly demonstrates the difference in running times as the values of  $n$  become large. This ratio is not as obvious when examining the individual graphs.

d. The type of curve that best fits each data set:

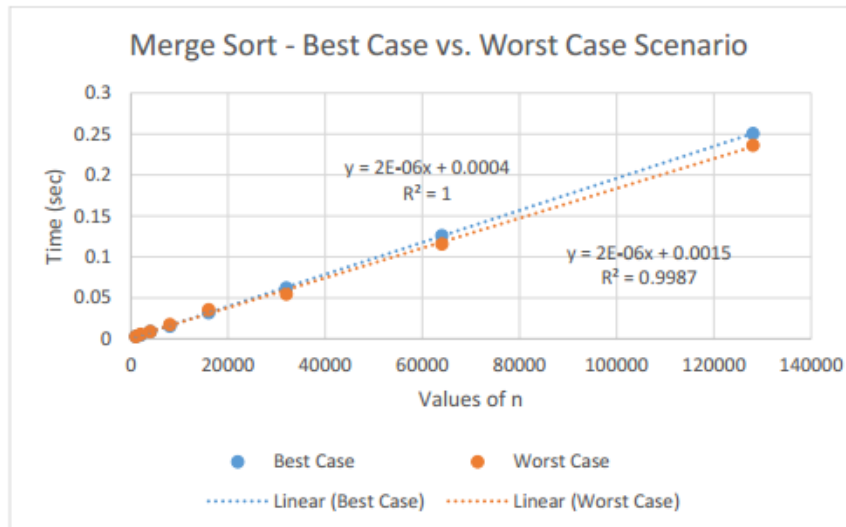
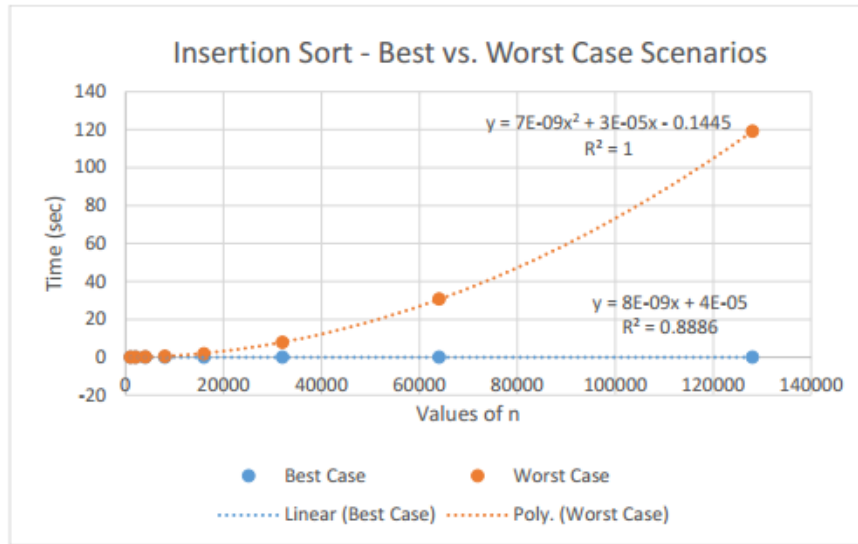
- Both data sets adhere closely to their predicted curves.
- Insertion sort behaves as a 2<sup>nd</sup> degree polynomial with  $y = 4e-09x^2 - 2e-06x + 0.0039$
- The  $R^2$  value of the insertion sort data is 1, indicating a perfect trendline, most likely a result of finding the average of three runs
- The merge sort behaves like a linear function, as  $n$  dominates  $\log n$ , with  $y = 2e-06x + 0.0004$
- The  $R^2$  value of the merge sort data was also very close at 0.9999, again most likely as result of finding the average of three runs.

e. How do the experimental running times compare to the theoretical running times of the data?

- The experimental running times of data are exactly on par with the theoretical running times for both methods. This is evidenced by the  $R^2$  value of 1 for the insertion sort data and the almost perfect  $R^2$  value of 0.9999 for the merge sort data. In both cases, the data behaved exactly as predicted with insertion sort adhering to  $O(n^2)$  and merge sort adhering to  $O(n \log n)$ . It is interesting to note that, although merge sort has a  $O(n \log n)$  it behaves more linear than  $n \log n$  on the graph. This is due to  $n$  being dominant over the values of  $\log n$ , but those values of  $\log n$  could be what is preventing the  $R^2$  value from achieving that perfect value of 1.



## CS 325 - HW 1 Solutions Examples



Running each algorithm under best case and worst case scenarios lead to some interesting results. In examining the results for Insertion Sort, there is a huge variance between best case (linear) and worst case scenario (2<sup>nd</sup> degree polynomial). The running times for the best scenario were essentially at zero, but the worst case scenario, while a 2<sup>nd</sup> degree polynomial like the average insertion sort scenario, took considerably more time than the average case. This is in stark contrast to the merge sort algorithm, which remained steady regardless of the scenario. This leads me to believe that some algorithms are more volatile than others, in that there are significant differences depending on the input scenario, while others like merge sort are more stable and return predictable results regardless of the input. This provides evidence of how important it is to consider the type of input your algorithms will be receiving. If you have an array that you know will be mostly sorted, insertion sort is the way to go; however, if that won't always be the case, or if there will be large values of n, an algorithm like merge sort that provides a more consistent running time would be the better option.