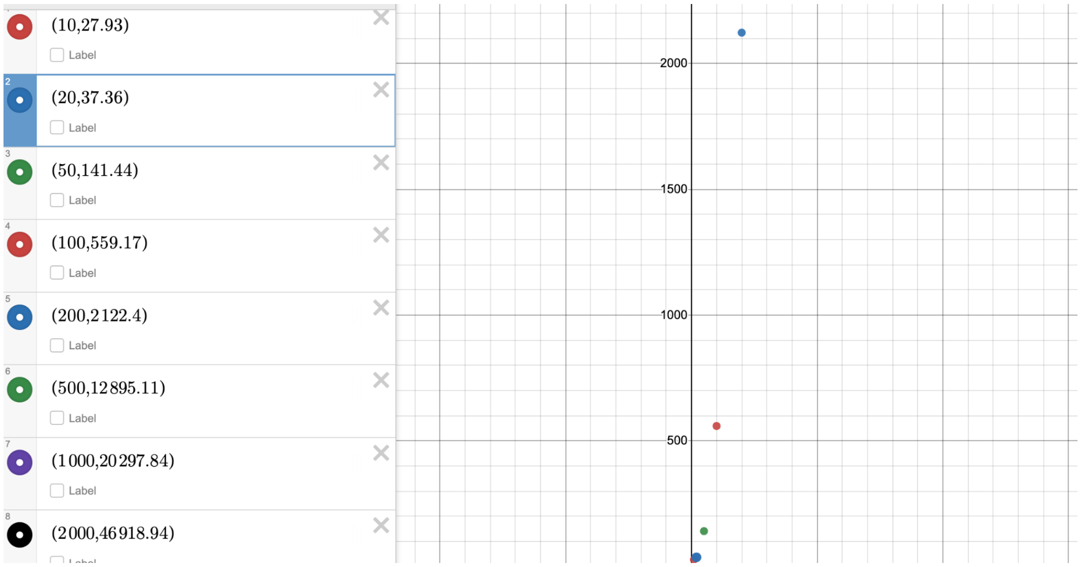
Jaap Singh
CSDS 233
Programming Assignment 5

Questions and Answers

Insertion Sort:



1 (10,27.93) □ Label

2 (20,37.36) □ Label

3 (50,141.44) □ Label

4 (100,559.17) □ Label

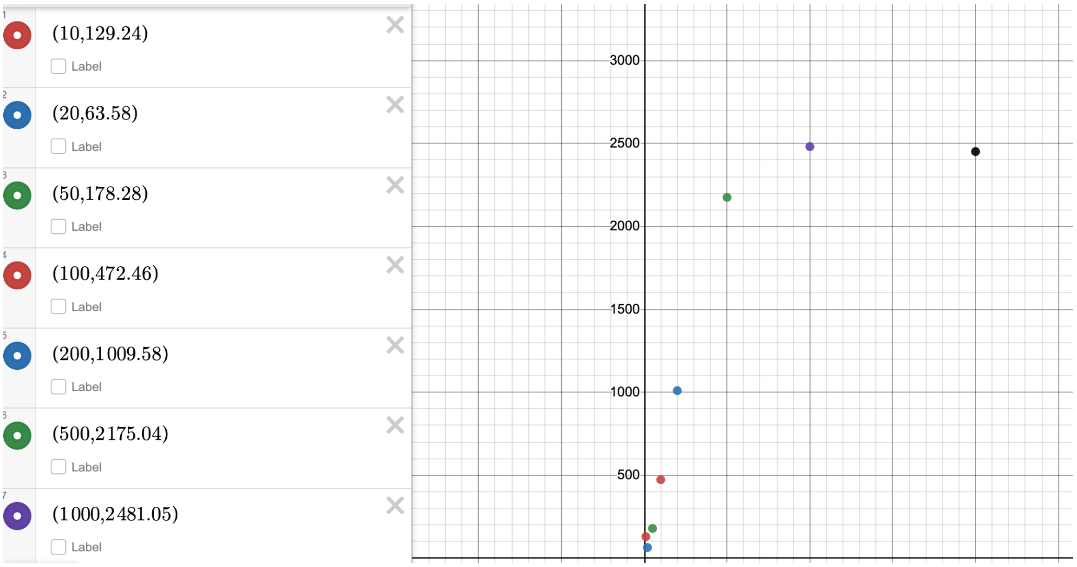
5 (200,2122.4) □ Label

6 (500,12895.11) □ Label

7 (1000,20297.84) □ Label

8 (2000,46918.94) □ Label

Quick Sort:



1 (10,129.24) □ Label

2 (20,63.58) □ Label

3 (50,178.28) □ Label

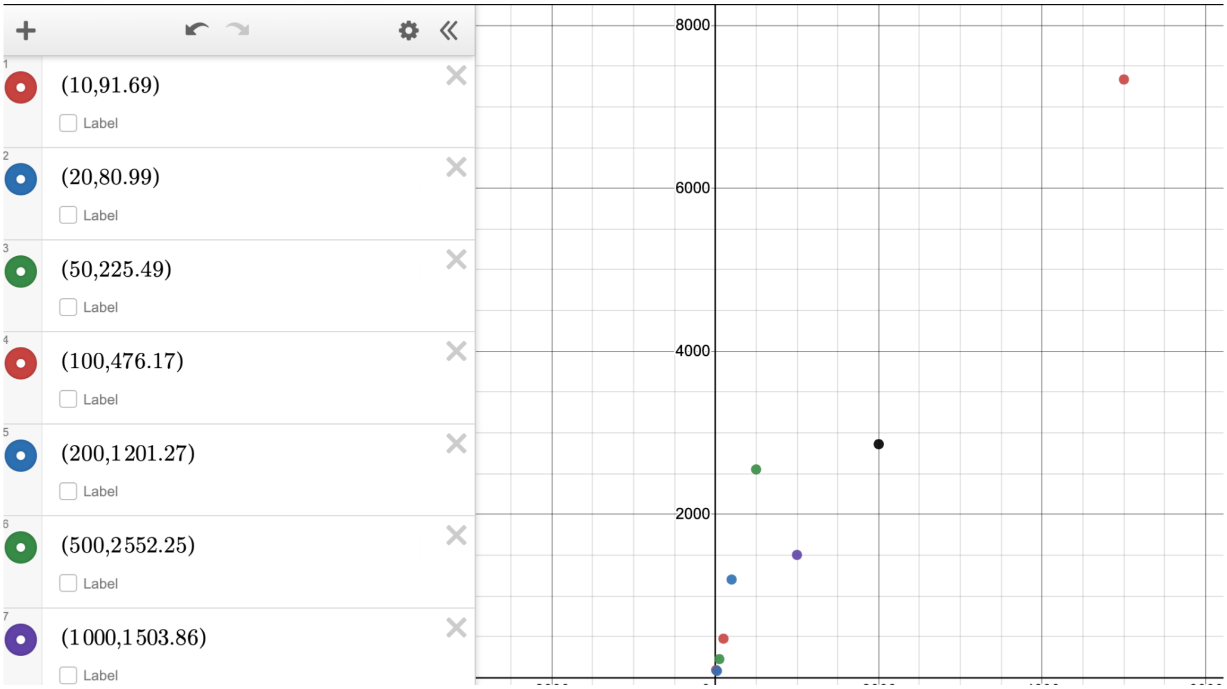
4 (100,472.46) □ Label

5 (200,1009.58) □ Label

6 (500,2175.04) □ Label

7 (1000,2481.05) □ Label

Merge Sort:


1 (10,91.69)

2 (20,80.99)

3 (50,225.49)

4 (100,476.17)

5 (200,1 201.27)

6 (500,2 552.25)

7 (1 000,1 503.86)

1. I ran the tests on the Eclipse IDE using Java 11. The input was all random numbers, so in situations in which the numbers were more sorted, the runtime would be lower.

2. The fastest method for large n is quicksort. Insertion sort follows an exponential curve; quicksort follows a logarithmic curve; merge sort follows a somewhat linear curve (if we draw a line of best fit).

3. Yes, my answer is consistent with my theory. If we look at the graphs, we can see that for large inputs (n), the graph for quicksort begins to plateau and stay lower than the other two graphs.

4. Insertion sort is preferable when the input is small. If we look at graphs, we can see that for small inputs, the runtime for insertion sort is lower than the other 2 graphs.

5. Insertion sort performs poorly with large n. If we look at the graph, we can see that the runtime is insanely large when large inputs are passed in.