Leah Benitez

Professor Fakhouri

CMP 338

September 22, 2020

Homework 2 Analysis

**Results**:

BubbleSort Equal 1000

-------------------------------------------------------------------------------------------------------------------------

16845675 4613420 6159481 530146 455507 431901 430678 431278 419036 399225 --- 3071634.7

BubbleSort Random 1000

-------------------------------------------------------------------------------------------------------------------------

14207081 1954166 1818113 868180 832485 831160 834015 831299 834587 881810 --- 2389289.6

BubbleSort Increasing 1000

-------------------------------------------------------------------------------------------------------------------------

1168064 722927 629429 626208 626425 628713 779944 727780 627818 626412 --- 716372.0

BubbleSort Decreasing 1000

-------------------------------------------------------------------------------------------------------------------------

8098568 9227460 5303749 3574530 1663035 1750267 1491420 2693723 2710635 1827139 --- 3834052.6

BubbleSort IncreasingAndRandom 1000

-------------------------------------------------------------------------------------------------------------------------

1800835 1552314 1576796 1545307 1563265 1792835 2054956 1654940 1518423 1687655 --- 1674732.6

SelectionSort Equal 1000

-------------------------------------------------------------------------------------------------------------------------

9617426 4082642 1346498 3856240 3739320 230455 227718 227224 227039 226312 --- 2378087.4

SelectionSort Random 1000

-------------------------------------------------------------------------------------------------------------------------

22518960 494963 452891 436783 438803 455743 437797 436865 437802 437906 --- 2654851.3

SelectionSort Increasing 1000

-------------------------------------------------------------------------------------------------------------------------

450069 461121 438316 571785 438015 437209 435444 437290 435461 437287 --- 454199.7

SelectionSort Decreasing 1000

-------------------------------------------------------------------------------------------------------------------------

549102 512775 441105 556724 681641 1053115 1009295 588800 578714 541237 --- 651250.8

SelectionSort IncreasingAndRandom 1000

-------------------------------------------------------------------------------------------------------------------------

4752841 449517 470200 439491 436068 436280 438089 436482 437842 436383 --- 873319.3

InsertionSort Equal 1000

-------------------------------------------------------------------------------------------------------------------------

2698053 188386 143918 75950 79530 66702 67482 69048 160844 124852 --- 367476.5

InsertionSort Random 1000

-------------------------------------------------------------------------------------------------------------------------

14370238 18728 20280 10907 11939 14561 10884 10852 11584 10838 --- 1449081.1

InsertionSort Increasing 1000

-------------------------------------------------------------------------------------------------------------------------

12640 11796 36415 18959 17405 19578 20956 23393 18484 20997 --- 20062.3

InsertionSort Decreasing 1000

-------------------------------------------------------------------------------------------------------------------------

13308762 19108 13207 12215 51083 14180 12361 12192 30124 12655 --- 1348588.7

InsertionSort IncreasingAndRandom 1000

-------------------------------------------------------------------------------------------------------------------------

88509 45659 12061 11715 11754 27708 11852 11703 11708 11693 --- 24436.2

BubbleSort Equal 10000

-------------------------------------------------------------------------------------------------------------------------

192242038 175646171 193865584 177980838 224958268 175409899 191898697 211507849 207405721 247818682 --- 1.998733747E8

BubbleSort Random 10000

-------------------------------------------------------------------------------------------------------------------------

663959116 349946053 313954939 313043279 294682409 375358432 327621023 299438063 353607673 326777585 --- -6.76578724E7

BubbleSort Increasing 10000

-------------------------------------------------------------------------------------------------------------------------

239512041 178445461 175262898 174686185 199698592 189534881 208449062 232527950 243994513 194983472 --- 2.037095055E8

BubbleSort Decreasing 10000

-------------------------------------------------------------------------------------------------------------------------

582129485 455321955 459789224 487044704 512687731 545082833 438655999 490270221 443681213 717722484 --- 8.37418553E7

BubbleSort IncreasingAndRandom 10000

-------------------------------------------------------------------------------------------------------------------------

223100218 244691164 232896203 205502784 199647370 191213130 178592766 187515942 199139191 523370528 --- -1.909298E8

SelectionSort Equal 10000

-------------------------------------------------------------------------------------------------------------------------

61012845 58135098 48177857 55771752 64290454 51320002 46075560 47602265 44599250 47236920 --- 5.24222003E7

SelectionSort Random 10000

-------------------------------------------------------------------------------------------------------------------------

62076310 77706172 51893926 52900819 51261125 50882945 66739645 83990993 53711231 47052954 --- 5.9821612E7

SelectionSort Increasing 10000

-------------------------------------------------------------------------------------------------------------------------

50319855 53586451 55630155 48641881 51108431 47651416 49526994 49272886 47840428 57046663 --- 5.1062516E7

SelectionSort Decreasing 10000

-------------------------------------------------------------------------------------------------------------------------

54816719 53674467 57452842 51323938 47048885 51522140 50795669 55586776 48461698 48070516 --- 5.1875365E7

SelectionSort IncreasingAndRandom 10000

-------------------------------------------------------------------------------------------------------------------------

53134306 50539258 56328835 49906046 64647110 46887841 46249324 47502882 50741244 47643317 --- 5.13580163E7

InsertionSort Equal 10000

-------------------------------------------------------------------------------------------------------------------------

49308 22469 22352 22380 22412 49896 35682 38800 36576 40468 --- 34034.3

InsertionSort Random 10000

-------------------------------------------------------------------------------------------------------------------------

333706824 88232 77969 77652 78033 78052 77639 78055 101686 91571 --- 3.34455713E7

InsertionSort Increasing 10000

-------------------------------------------------------------------------------------------------------------------------

44978 46786 45960 43165 45530 44391 44393 42423 41195 41426 --- 44024.7

InsertionSort Decreasing 10000

-------------------------------------------------------------------------------------------------------------------------

373603502 46788 31748 31446 31344 31468 31378 31452 31258 31244 --- 3.73901628E7

InsertionSort IncreasingAndRandom 10000

-------------------------------------------------------------------------------------------------------------------------

2622039 38852 24735 24357 24340 24405 24286 24319 24343 24357 --- 285603.3

**Analysis**:

Using the test time tables you created by running Driver.main(), copy your results into a Microsoft Word document and answer the following questions using 1-3 complete sentences for each question:

1. **Create graphs from the test times you measured.**
2. **Which sort worked best on data in constant or increasing order (ie already sorted data)?**

The InsertionSort worked best on data in constant or increasing order.

1. **Why do you think this sort worked best?**

For InsertionSort, the inner loop iterates backwards over the *sorted* part of the array and makes any swaps necessary to ensure the current element (controlled by the outer loop) ends up at the right place. Once it is in the right place, the inner loop terminates. Since the arrays with data in constant or increasing order are already sorted, the inner loop gets to terminate after the first comparison.

For SelectionSort, the inner loop iterates forwards over the *unsorted* part of the array, so every element must be searched through in order to find an accurate minimum for each swap. This means the inner loop cannot terminate early like in the case of InsertionSort, which increases runtime. \*Note: Even though the arrays in question are already sorted, the algorithm does not know that, and it will still separate the data into a sorted and unsorted part.

For BubbleSort, the array is iterated through n – 1 times, where n = the number of elements in the array. For each iteration, every element is compared at some point. So even though the arrays in question are sorted and no swaps will happen, there will be a lot of iterations and comparisons, which increases runtime.

1. **Did the same sort do well on the case of mostly sorted data? Why or why not?**

Yes, InsertionSort still did well in the case of mostly sorted data. In this case, the same point as before still stands: for a large portion of the data, the inner loop got to terminate after the first comparison. And even for the unsorted portion of the data, the loop got to terminate once the right place was found. It did not have to go through the whole sorted portion of the array (with the exception of the worst case).

1. **In general, did the ordering of the incoming data affect the performance of the sorting algorithms? Please answer this question by referencing specific data from your table to support your answer.**

All of the algorithms seemed to perform best for increasing data, which you can see has the lowest test times in the average graphs. You can also see that the values are smaller in the Increasing 1000 and Increasing 10000 graphs compared to the values in the other graphs (with the exception for SelectionSort for arraySize = 10000)

BubbleSort struggled with decreasing values, which you can see in the average graph (namely for arraySize = 10000). I believe this has to do with the amount of iterations and swaps especially that had to happen.

The ordering of the data did not really seem to matter for SelectionSort when arraySize = 10000. The averages seem to stay the same and in the regular graphs, you can see that the test times stayed at 50,000,000. The ordering of the data mattered when arraySize = 1000. The algorithm struggled with equal and random values and I cannot really say why.

InsertionSort struggled with random and decreasing values, but only for arraySize = 10000. In the average graphs for arraySize = 1000, it looks like the algorithm struggled, but this was only because of outliers. Really, the test times stayed at the lower 100,000 nanoseconds range.

1. **Which sort did best on the shorter (ie n=1,000) data sets? Did the same one do better on the longer (ie n=10,000) data sets? Why or why not? Please use specific data from your table to support your answer.**

InsertionSort seemed to do best for shorter and longer data sets with the exception of decreasing data values when n = 1,000. In that case, SelectionSort did better. I believe this is because the inner loop for InsertionSort had to work harder to get the smaller values to the left of the sorted portion of the array. It did not get to terminate early.

I think InsertionSort generally does better because of the fact that the inner loop can terminate early.

1. **In general, which sort did better? Give a hypothesis as to why the difference in performance exists.**

Like I said before, InsertionSort generally did better. I think this is because the inner loop gets to terminate once the element being evaluated finds the right place. The inner loop for SelectionSort does not get to terminate early, and the array for BubbleSort has to iterate n – 1 times, with n = length of array.

1. **Are there results in your table that seem to be inconsistent? (ex. If I get run times for a sort that look like this {1.3, 1.5, 1.6, 7.0, 1.2, 1.6, 1.4, 1.8, 2.0, 1.5] the 7.0 entry is not consistent with the rest). Why do you think this happened?**

There are results that seem to be inconsistent, which you can see in pretty much every graph. This often happens the very first time a method is called. We have not learned about this in class, and I have not learned about it in any other class. But my personal theory is that it has something to do with the system memory (or the memory in Eclipse). Maybe once a pathway is made, it stores it, and makes future tests go faster.