DSA I Lab 6: Sorts

Christine Baca [cab8xd]

1 University of Virginia, Charlottesville VA 22904, USA

**Abstract.** The experiment investigated implementation and runtimes of the generic-type and Comparable-inherited sorting algorithms bubbleSort, insertSort, mergeSort, and quickSort through four experiments that emphasized. The lab concludes that, unless the data set involves numbers that are “almost-sorted” or the threshold of disorderliness is exceptionally low, the quick sort is the fastest sort. Otherwise, insertion sort is the fastest sort.

**Keywords:** bubbleSort, mergeSort, quickSort, insertSort, Java, Lists

The lab explored the efficiency of different sorts in Java through testing the sorts in four experiments.

* 1. Experiment One

Experiment One runs an array with a set of random values through each sort method and is times. Quicksort or mergeSort is expected to run the fastest due to their big theta *nlog(n)* runtimes.

//Experiment One Example Test Code:

//BUBBLE SORT TESTING

//Starting timer

t = (**int**) System.*currentTimeMillis*();

//Sorting list

*bubbleSort*(bub);

//Recording Time

time = ((**int**) System.*currentTimeMillis*()) - t;

b.add(time);

* 1. Experiment Two

Experiment Two runs through same procedure as the first experiment except the lists generated are sorted reversely. The same results are expected.

* 1. Experiment Three

Experiment three runs each list through an algorithm that’s almost sorted. The same results are expected although there is the possibility of insertion sort running faster in this specific experiment due to its ability to adjust small errors in unsorted lists more quickly than the other sorts.

* 1. Experiment Four

Experiment Four implements a hybrid sort that switches algorithms mid experiment to optimize runtimes based on the results of the previous experiments. The hybrid sort takes the size of the array left and deduces which sort to use.

//Experiment Four

**private** **static** <T **extends** Comparable<T>> **void** hybridSort(T[] list){

*hybridSort*(list, 0, list.length - 1);

}

**private** **static** <T **extends** Comparable<T>> **void** hybridSort(T[] list, **int** i, **int** j){

**if**(j - i < 1000)

{

*insertionSort*(list);

}

**else**

{

*quickSortH*(list, i , j);

}

}

1. Experiment:

The procedure for the experiment runs through numerous trials of that sort methods using various data sets, recording the runtime of the sorts. The average and standard deviation of the trials will be calculated and recorded. The data will be analyzed based on the trial’s runtimes, averages, and standard deviations. The experiment table lists the number of trials, runs, and additional information regarding the experimental procedure.

* 1. Experiment Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Exp. 1** | **Exp. 2** | **Exp. 3** | **Exp. 4** |
| **No. of Trials** | 10 | 10 | 10 | 10 |
| **Number of Elements in List** | 10,000 | 10,000 | 10,000 | 10,000 |

1. Results
   1. Table Results

Experiment One:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trials | bubbleSort | insertionSort | mergeSort | quickSort |
| 1 | 525 | 148 | 213 | 6 |
| 2 | 314 | 113 | 139 | 18 |
| 3 | 273 | 65 | 39 | 2 |
| 4 | 257 | 67 | 140 | 1 |
| 5 | 266 | 68 | 38 | 1 |
| 6 | 275 | 63 | 38 | 2 |
| 7 | 347 | 64 | 317 | 2 |
| 8 | 261 | 92 | 36 | 1 |
| 9 | 299 | 67 | 50 | 2 |
| 10 | 256 | 63 | 21 | 1 |
| Average | 307.3 | 81 | 103.1 | 2 |
| Standard Deviation | 81.8630971 | 28.6511586 | 98.39201639 | 5.274677452 |
|  |  |  |  |  |

Experiment Two:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trials | bubbleSort | insertionSort | mergeSort | quickSort |
| 1 | 449 | 187 | 198 | 75 |
| 2 | 333 | 137 | 120 | 34 |
| 3 | 358 | 98 | 38 | 21 |
| 4 | 364 | 96 | 151 | 22 |
| 5 | 346 | 68 | 27 | 16 |
| 6 | 325 | 72 | 52 | 16 |
| 7 | 331 | 89 | 155 | 14 |
| 8 | 338 | 72 | 29 | 23 |
| 9 | 341 | 77 | 26 | 16 |
| 10 | 216 | 80 | 27 | 1 |
| Average | 340.1 | 97.6 | 82.3 | 2 |
| Standard Deviation | 56.3253446 | 37.33095231 | 66.51658106 | 19.82030384 |

Experiment Three:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Trials | bubbleSort | | | insertionSort | | mergeSort | quickSort |
| 1 | 353 | | | 176 | | 145 | 136 |
| 2 | 330 | | | 134 | | 131 | 96 |
| 3 | 169 | | | 69 | | 27 | 91 |
| 4 | 176 | | | 67 | | 117 | 92 |
| 5 | 178 | | | 73 | | 44 | 104 |
| 6 | 206 | | | 86 | | 166 | 112 |
| 7 | 214 | | | 86 | | 26 | 91 |
| 8 | 250 | | | 89 | | 33 | 93 |
| 9 | 178 | | | 99 | | 21 | 142 |
| 10 | 216 | | | 99 | | 40 | 99 |
| Average | 227 | | | 97.8 | | 75 | 105.6 |
| Standard Deviation | 65.41491505 | | | 33.62803064 | | 57.39918699 | 18.8514662 |
|  |  | | |  | |  |  |
| Experiment Four:  Trials | Exp. 1 | Exp. 2 | Exp. 2 | |
| 1 | 353 | 576 | 775 | |
| 2 | 330 | 634 | 731 | |
| 3 | 469 | 669 | 827 | |
| 4 | 476 | 767 | 617 | |
| 5 | 578 | 873 | 844 | |
| 6 | 106 | 386 | 666 | |
| 7 | 324 | 486 | 726 | |
| 8 | 420 | 589 | 533 | |
| 9 | 148 | 699 | 321 | |
| 10 | 566 | 522 | 330 | |
| Average | 447 | 550.4 | 603 | |

3.2 Graph Results

1. Conclusion

The experiment tested the runtimes of sorts through various experiments. The first experiment, implementing sorts on large lists of random integers, demonstrated quick sort as the fastest sort method and bubble sort as the slowest. While both the quick and bubble sorts having similar runtimes, quick sort shows to be better suited for smaller datasets. The second experiment is identical to the first but the data set given is in reverse sorted order. The results show that the quick sort ran the fastest. The third experiment is implemented the same as the previous two but with a data set that’s almost sorted. The results demonstrate that the quick sort ran the fastest. The fourth experiment creates a hybrid sort that attempts to move more quickly than the sorts already implemented by combining some of them into a new sort. The hybrid sort’s runtime was overall, slower than the sorts that comprise it. Errors include faulty coding and testing. Future experiments could implement known or other unmade hybrid sorts and compare their runtimes.