Xiaoshi Wang

CS 104 Lab

December 13, 2016

**Compare Sorting Algorithms**

**Hypothesis**

The order of four different sorting algorithms from fastest to slowest is Selection Sort, Insertion Sort, Merge Sort, and Quick Sort.

**Reasons**

Since the time for each step the computer runs is the same, we order the time spent for each sort by comparing their number of steps. Suppose there are n numbers. The number of steps for a selection sort is the largest. Since a selection sort selects the smallest number when it decides the value of each element. Many pairs of numbers are compared more than one time and plenty of repeated work could be reduced (as the other three selection methods).

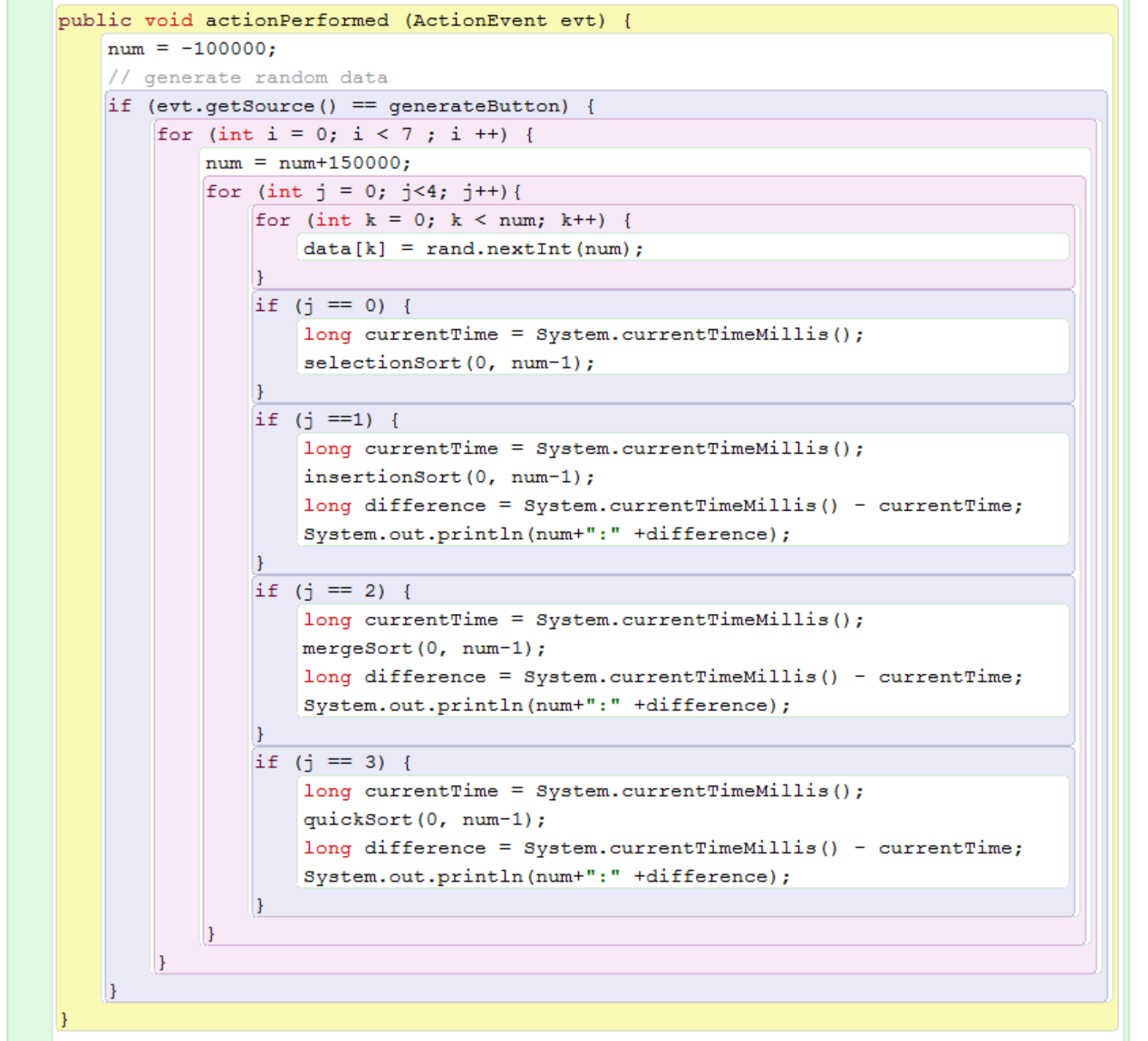
For the insertion Sort, it must take shorter time than the selection sort does because it does not compare an element with all the others. Instead, the comparing process stops when our number meets a number smaller than it in the processed part. The large for loop for comparison would be half shorter than that of the selection sort. But since we cannot actually insert our number into an array, the adjustment of the numbers after this one would takes a long time.

I think the time spent by the merge sort should be closed to that of the insertion sort. Since recursion is used in the merge sort to divide a big set of numbers into small pieces, we do not need to worry about the wasted rearrangement in the previous two methods any more. However, the merge part in merge sort looks a little bit complex. A certain number is merged again and again.

The quick sort should be the fastest. Since all the numbers are only compare with the pivot several times. The number of time should be the log n with base 2. This way need fewer steps than all the other three. Thus, this is the fastest.

**Process**

To find the time needed for the four different sorting, I changed the actionProformed method in Sorter and added a for-loop to print the time of each efficiently. The following screenshot is my new code.



We can see that a new array of data[k] is constructed by the for-loop of k. Also, we call different sort methods for different j value to go through all the four sorting for each length of the data array. To change the length of the data array, we add 150000 to variable num after we finish the last for-loop. Thus, when I press the generate Button, the code above would print the results for all the sorting methods with input of size 50000, 200000, 350000, 500000, 650000, 800000 and 950000. I process my experiment by pressing the generate Button five times.

**Data and Graphs**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data Size  Sort Method | **50000** | **200000** | **350000** | **500000** | **650000** | **800000** | **950000** |
| **Selection Sort** | 1015 | 15746 | 48304 | 98732 | 167959 | 258169 | 367405 |
|  | 1020 | 15590 | 37882 | 97565 | 163797 | 250586 | 251458 |
|  | 997 | 15463 | 47341 | 96631 | 164323 | 255189 | 386646 |
|  | 1007 | 15427 | 47301 | 96836 | 164672 | 256044 | 355627 |
|  | 1054 | 16650 | 49226 | 97585 | 173248 | 278148 | 380265 |
| Mean | **1018.6** | **15775.2** | **48010.8** | **97469.8** | **166799.8** | **259627.2** | **368280.2** |
| **Insertion Sort** | 451 | 7444 | 22738 | 46479 | 78551 | 119687 | 169957 |
|  | 457 | 7376 | 22451 | 45540 | 77082 | 324422 | 164165 |
|  | 467 | 7289 | 22283 | 45559 | 77351 | 121804 | 169502 |
|  | 460 | 7271 | 22401 | 45605 | 77595 | 117938 | 166809 |
|  | 477 | 7532 | 23078 | 46291 | 86822 | 125467 | 176030 |
| Mean | **462.4** | **7382.4** | **22590.2** | **45894.8** | **79480.2** | **162083.6** | **169292.6** |
| **Merge Sort** | 3 | 26 | 57 | 83 | 126 | 149 | 185 |
|  | 15 | 33 | 61 | 96 | 124 | 169 | 184 |
|  | 0 | 34 | 62 | 100 | 131 | 175 | 200 |
|  | 16 | 34 | 63 | 84 | 131 | 154 | 185 |
|  | 0 | 38 | 74 | 96 | 146 | 157 | 201 |
| Mean | **6.8** | **33** | **63.4** | **91.8** | **131.6** | **160.8** | **191** |
| **Quick Sort** | 0 | 16 | 22 | 47 | 76 | 100 | 105 |
|  | 0 | 31 | 31 | 47 | 63 | 85 | 100 |
|  | 0 | 16 | 38 | 54 | 63 | 93 | 101 |
|  | 16 | 31 | 38 | 53 | 63 | 85 | 100 |
|  | 0 | 20 | 47 | 53 | 69 | 100 | 116 |
| Mean | **3.2** | **22.8** | **35.2** | **50.8** | **66.8** | **92.6** | **104.4** |

Collected Data and Their Mean

Comparison of Four Sort Methods in Graph

Comparison of Merge Sort and Quick Sort

**Conclusion**

The graphs in the last part are constructed according to the means in the data table. From the graphs, we can clearly conclude that the order of time performance from long to short is: Selection Sort, Insertion Sort, Merge Sort and Quick Sort for all data sizes. Thus, this experiment reinforces our hypothesis.

The data array we constructed is generated by the random generator. However, some series of random numbers may use less time for a certain sort method than others. Thus, we may obtain a false conclusion. Thus, we processed five times for each Sort in each data size. Then, the probability of deducing a false conclusion would be reduced.