CSSE230: Sorting Races

# Name(s): Youheng Hou, Shuai Yuan

# Part 1: Data

Table 1 shows the runtimes of all 5 sorts for at least 4 different types of arrays:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2.0e+06 Integers | Random | Shuffled | Almost Sorted | Almost Reverse |
| Merge sort | 732ms | 581ms | 69ms | 78ms |
| JAVA quicksort | 294ms | 164ms | 40ms | 42ms |
| BST sort | 2435ms | 3462ms | 782ms | 753ms |
| Heap Sort | 1503ms | 1529ms | 421ms | 497ms |
| Quick sort | 234ms | 207ms | 49ms | 85ms |

# Part 2: Discussion

Include your discussion of the runtimes in Table 1, as described in the specification.

We have used the quick + insertion method to implement the quick sort.

The fastest for each is:

Random: my quick sort

Random no dup: JAVA quick sort

Almost: JAVA quick sort

Reverse Almost: JAVA quick sort

This result is not very surprising. Since JAVA’s quick sort probably have a different cut off point of insertion sort than our code and more pivots, the run time for JAVA’s is different. In addition, JAVA probably had done some research on the cutoff point, whereas we just picked whatever we felt like. Merge sort being the 2-3 place is not surprising either since the run time is pretty linear due to the separation. Lastly, bst and heap sort takes last two place because it takes time to populate the tree/heap.