CSSE230: Sorting Races

# Name(s): Zachary Taylor

# Part 1: Data

Table 1 shows the runtimes of 6 sorts for at least 4 different types of arrays:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Random | Shuffled | Almost Sorted | Almost(reverse) Sorted |
| Default 1 | 109ms | 47ms | 47ms | 62ms |
| Default 2 | 312ms | 172ms | 78ms | 78ms |
| Tree Set | 1950ms | 1185ms | 593ms | 546ms |
| Quick Sort | 266ms | 95ms | 156ms | 156ms |
| BinaryHeap Sort |  |  |  |  |

# Part 2: Discussion

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

Default 1: The default implementation of int is to use merge sort. This preformed as expected because it is a default sort.

Default 2: The default implementation of Integer is to use quick sort, this preformed as expected in it’s best case and worst case.

Tree set: This preformed pretty poorly. It might be a fault of my implementation. It is supposed to run in O(n\*log(n)) time which would mean that it shouldn’t stray to be as far away as it is from other O(n) sorts.

Quick Sort: This was a pretty quick sort. In general it ran pretty well compared to the other sorts implemented. It seemed to follow the general trend of Default 2 (which uses quick sort) while being slower in almost all cases.

Memory issues:

There were no memory issues in the final versions of the programs. There were memory issues with quick sort’s added insertion sort.

The fastest for each type was as follows:

Default 1 for everything.

Surprising:

Quick sort is pretty difficult to implement entirely.