CSSE230: Sorting Races

# Name(s): Aidan Kirk

# Part 1: Data

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

Size = 10,000,000

Max value = Integer.MAX\_VALUE

Time is entered in milliseconds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Default using ints | Defaults using Integers | TreeSort usingIntegers | QuickSort using ints | HeapSort using Integers |
| Random Array | 747 | 5450 | 16217 | 909 | 14487 |
| Shuffled Array | 665 | 5102 | 15781 | 940 | 13488 |
| Almost Sorted Array | 263 | 380 | 2372 | 160 | 3401 |
| Reverse Almost Sorted Array | 227 | 420 | 2383 | 312 | 3782 |

# Part 2: Discussion

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

All of the sorting algorithms ran fine or any array of size up to 10,000,000. If I added an extra zero, I would sometimes get a heap out of space error from java during the first default sort. Other than that, there were no memory issues. One notable thing is that after running the test that produced the values in my table, I ran the test again, but with the max value as the size of the array. The only thing that changed was that tree sort of Inteers was twice as fast. Thus, it must be that tree sorting is much faster when sorting duplicates.

For the random array, the default sort on ints was the quickest. This is most likely because random arrays are unpredictable. Thus, the divide and conquer algorithm with 2 pivots really helps eliminate the randomness early to get to the sorted array quicker.

For the shuffled array, the default sort for ints was the quickest. This is most likely due to a similar reason to the previous answer.

For the almost sorted array, the race was much closer, and the quick sort I wrote was the fastest. Thus, this leads me to believe that a 1-pivot quick sort is faster for sorting arrays that are almost sorted. This was also the first case where heap sort was slower than tree sort. Thus, tree sort is better for sorting almost-sorted arrays.

Finally, the almost sorted reverse array. The default int quick sort went back to being the fastest. However, the results were very similar to the previous row in the table.

I was surprised that the 1-pivot quick sort was faster than the default quick sort at all. It was interesting that an almost sorted array led to a quicker time for my quick sort than java’s. It must be that more unnecessary work is being done in the default quick sort in this case.