Lamess Kharfan

Student ID: 10150607

CPSC 331 – Assignment 4

6. A5Q5.java tests the four sorting algorithms and Java's built in sort function on randomly generated arrays of various lengths. The VisualVM Profiler was used to determine total runtime, in milliseconds, required to sort arrays of the following lengths:

	128	1024	16384	65536
Insertion Sort	10.2	199	38, 265	611, 439
Heap Sort	27.9	165	2313	9969
Quick Sort	24.3	175	1839	7744
Quick Sort Improved	3.76	157	1817	8191
Quick Sort Bonus	29.2	161	2711	11, 620
Java's Built-in Sorting Function	48.4	96.1	1096	4119

These results are as I would expect, for very small arrays insertion sort is the most efficient, but for very large randomly generated arrays, insertion sort is by far the least efficient since it has to sort progressively larger subarrays, and when the array is very larger, this is an extremely lengthy process. Quick Sort/Quick Sort Improved are the best case for all lengths of arrays when they're randomly generated, as expected since it's a recursive "Divide and Conquer" sorting algorithm. The total runtimes for quick sort and quick sort improved are very similar for large arrays because they both use the recursive algorithm, however, for smaller arrays, quick sort improved is faster because it uses insertion sort. We see that quick sort improved is slightly slower for larger arrays, this is likely due to the extra checking of the size of the array to decide if we should use insertion sort, or sort recursively. Heap sort requires insertions and deletions, so we can expect it to be slower than quick sort, but much faster than insertion sort on average. Overall, the results are as we would expect them.

6. A5Q6.java tests the four sorting algorithms and Java's built in sort function on sorted arrays of various lengths. The VisualVM Profiler was used to determine total runtime, in milliseconds, required to sort arrays of the following lengths:

	128	1024	16384
Insertion Sort	3.50	5.48	61.5
Heap Sort	67.8	280	3385
Quick Sort	41.5	428	50, 248
Quick Sort Improved	0.678	1.0	2471
Quick Sort Bonus	82.2	182	2610
Java's built in sorting function	24.2	43.0	57.8

The results are in fact as we would expect them, with insertion sort running the most efficiently and quick sort running the least efficiently. The runs times of quicksort improved shows that when arrays are of a smaller size, insertion sort is being used and when they are of a larger size, the array is recursively sorted, which helps the algorithm run much more efficiently than normal quicksort. Quicksort's worst case run time occurs when the array is already sorted. However, for insertion sort, when the array is already sorted, this is its best-case since it simply does a constant amount of work. For heap sort, in an already sorted array, its total runtime is quite a bit faster than it is for a randomly generated array, which can be expected since it needs to do a lot less work.

Bonus Question:

The version of quick sort that has worst case runtime O(n log n) is randomized quicksort. The only modification to quicksort made is that the pivot element is chosen randomly, within the bounds of the array, and everything else is implemented the same. The total runtimes for sorting 1000 randomly generated arrays and 1000 sorted arrays is presented above. The runtimes are as expected, since only in the worst case would we expect to see the run times being in O(n log n), the above shows about an average run time.