Evan Heaton : CS315 : Homework 8 BONUS : 12/2/15

1. Upon running the program, how to use it should be self-explanatory. After you are done with the first section, which enters arrays from the keyboard, press 3 to continue to the large array tests.

2. On arrays that were in descending order, insertion sort failed miserably. This was to be expected, though, because this is the case in which it runs in $O(n^2)$ time. Quick sort lived up to its name and ran very quickly, and this was to be expected as well. This particular implementation of quick sort choosing random pivots should fare well in almost any ordering of input. Merge sort was the middle ground, with a decent amount more comparisons than quick sort.

3. On arrays that were in the wacky, mostly ascending order, insertion sort did much better. Insertion sort runs its fastest on nearly sorted arrays, and that property was made very clear through this testing. Merge sort performed AWFULLY on the nearly sorted array; for shame, merge sort. As for quick sort, well, it's still pretty quick. Quick sort seems to be a very good choice of sorting algorithm, because it handles so many types of input so well.

Here's the output to running my code:

which sorting algorithm:
0. insertion
1. merge
2. quick
3. quit (continue to large arrays)
test
invalid input
which sorting algorithm:
0. insertion
1. merge
2. quick
3. quit (continue to large arrays)
2
enter whitespace delimited integers, q to quit
1 3 4 6 8 9 7 2 5 q
<1, 3, 4, 6, 8, 9, 7, 2, 5>
needed 12 comparisons for quick sort
<1, 2, 3, 4, 5, 6, 7, 8, 9>
which sorting algorithm:
0. insertion
1. merge
2. quick
3. quit (continue to large arrays)

3

descending order:
n = 100
needed 5049 comparisons for insertion sort
needed 356 comparisons for merge sort
needed 112 comparisons for quick sort
n = 200
needed 20099 comparisons for insertion sort
needed 1168 comparisons for merge sort
needed 232 comparisons for quick sort
n = 400
needed 80199 comparisons for insertion sort
needed 2992 comparisons for merge sort
needed 451 comparisons for quick sort
n = 800
needed 320399 comparisons for insertion sort
needed 7040 comparisons for merge sort
needed 904 comparisons for quick sort
n = 1600
needed 1280799 comparisons for insertion sort
needed 15936 comparisons for merge sort
needed 1811 comparisons for quick sort
n = 3200
needed 5121599 comparisons for insertion sort
needed 35328 comparisons for merge sort
needed 3622 comparisons for quick sort
n = 6400
needed 20483199 comparisons for insertion sort
needed 77312 comparisons for merge sort
needed 7221 comparisons for quick sort

press enter to continue to wacky order...

wacky order:
<6, 5, 4, 3, 2, 1, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 100, 99, 98>
n = 100
needed 117 comparisons for insertion sort
needed 77631 comparisons for merge sort
needed 64 comparisons for quick sort
n = 200

needed 217 comparisons for insertion sort
needed 78366 comparisons for merge sort
needed 128 comparisons for quick sort
n = 400
needed 417 comparisons for insertion sort
needed 80033 comparisons for merge sort
needed 265 comparisons for quick sort
n = 800
needed 817 comparisons for insertion sort
needed 83764 comparisons for merge sort
needed 511 comparisons for quick sort
n = 1600
needed 1617 comparisons for insertion sort
needed 92023 comparisons for merge sort
needed 1011 comparisons for quick sort
n = 3200
needed 3217 comparisons for insertion sort
needed 110138 comparisons for merge sort
needed 2029 comparisons for quick sort
n = 6400
needed 6417 comparisons for insertion sort
needed 149565 comparisons for merge sort
needed 4046 comparisons for quick sort
Program ended with exit code: 0