Darryl Green

CIS 2353

Winter 2018

Baugh

TestingSort results:

```
run:
Size: 10
Elapsed time for QuickSort after 5 runs in nanoseconds: 8193
Elapsed time for InsertionSort after 5 runs in nanoseconds: 3218
InsertionSort is faster in this case.

Size: 25
Elapsed time for QuickSort after 5 runs in nanoseconds: 18724
Elapsed time for InsertionSort after 5 runs in nanoseconds: 5851
Insertion Sort is faster in this case.

Size: 50
Elapsed time for QuickSort after 5 runs in nanoseconds: 47104
Elapsed time for InsertionSort after 5 runs in nanoseconds: 18725
Insertion Sort is faster in this case.

Size: 100
Elapsed time for QuickSort after 5 runs in nanoseconds: 56467
Elapsed time for InsertionSort after 5 runs in nanoseconds: 72264
Quicksort is faster in this case.

Size: 200
Elapsed time for QuickSort after 5 runs in nanoseconds: 52663
Elapsed time for InsertionSort after 5 runs in nanoseconds: 201289
Quicksort is faster in this case.

Size: 400
Elapsed time for QuickSort after 5 runs in nanoseconds: 121417
Elapsed time for InsertionSort after 5 runs in nanoseconds: 1651272
Quicksort is faster in this case.

Size: 800
Elapsed time for QuickSort after 5 runs in nanoseconds: 182859
Elapsed time for InsertionSort after 5 runs in nanoseconds: 2918107
Quicksort is faster in this case.

The total elapsed time for quicksort for all cases: 487427
The total elapsed time for insertion sort for all cases: 4870726
As the list size continues to grow QuickSort is faster overall for all test cases.
BUILD SUCCESSFUL (total time: 0 seconds)
```

The reason we get the results that are shown above is because as the size of the array increased from 50 to 100, 200, 400, 800, etc. it became more efficient and faster to utilize the QuickSort method versus

InsertionSort which works better on smaller number sets such as 10, 25, 50, 100, etc. before reaching a set of numbers greater than 100.

As shown below in the second set of outputs this time Set 100 ran faster with the InsertionSort method. What I think is happening here is that 100 will at times run better with InsertionSort and others with QuickSort because it is just small enough that one method will at time run better than the other. Though, once we start using integer sets greater than 100 it becomes consistently apparent that QuickSort is the more efficient sorting method to use when working with such large number sets such as 200, 400, 800 etc.

```
run:
Size: 10
Elapsed time for QuickSort after 5 runs in nanoseconds: 8193
Elapsed time for InsertionSort after 5 runs in nanoseconds: 3218
InsertionSort is faster in this case.

Size: 25
Elapsed time for QuickSort after 5 runs in nanoseconds: 19017
Elapsed time for InsertionSort after 5 runs in nanoseconds: 6143
Insertion Sort is faster in this case.

Size: 50
Elapsed time for QuickSort after 5 runs in nanoseconds: 46226
Elapsed time for InsertionSort after 5 runs in nanoseconds: 19310
Insertion Sort is faster in this case.

Size: 100
Elapsed time for QuickSort after 5 runs in nanoseconds: 59098
Elapsed time for InsertionSort after 5 runs in nanoseconds: 49737
Insertion Sort is faster in this case.

Size: 200
Elapsed time for QuickSort after 5 runs in nanoseconds: 55588
Elapsed time for InsertionSort after 5 runs in nanoseconds: 180809
Quicksort is faster in this case.

Size: 400
Elapsed time for QuickSort after 5 runs in nanoseconds: 123465
Elapsed time for InsertionSort after 5 runs in nanoseconds: 866889
Quicksort is faster in this case.

Size: 800
Elapsed time for QuickSort after 5 runs in nanoseconds: 192219
Elapsed time for InsertionSort after 5 runs in nanoseconds: 682862
Quicksort is faster in this case.

The total elapsed time for QuickSort for all cases: 503806
The total elapsed time for InsertionSort for all cases: 1808968
As the list size continues to grow QuickSort is shown to be faster with larger test cases.
BUILD SUCCESSFUL (total time: 0 seconds)
```