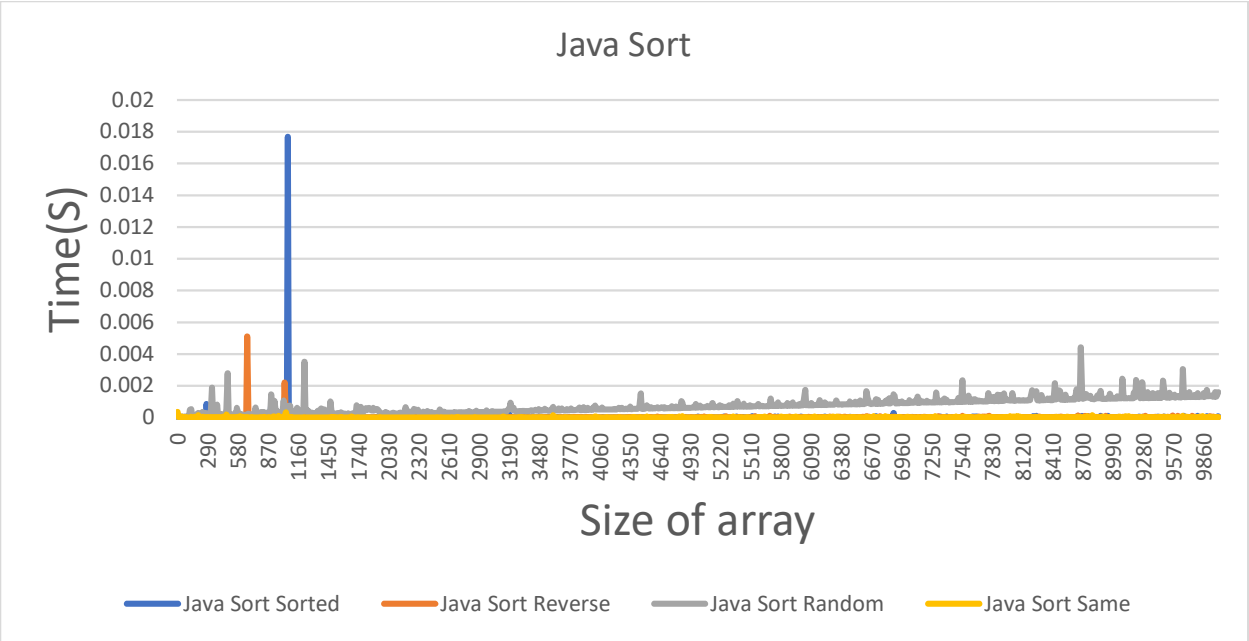
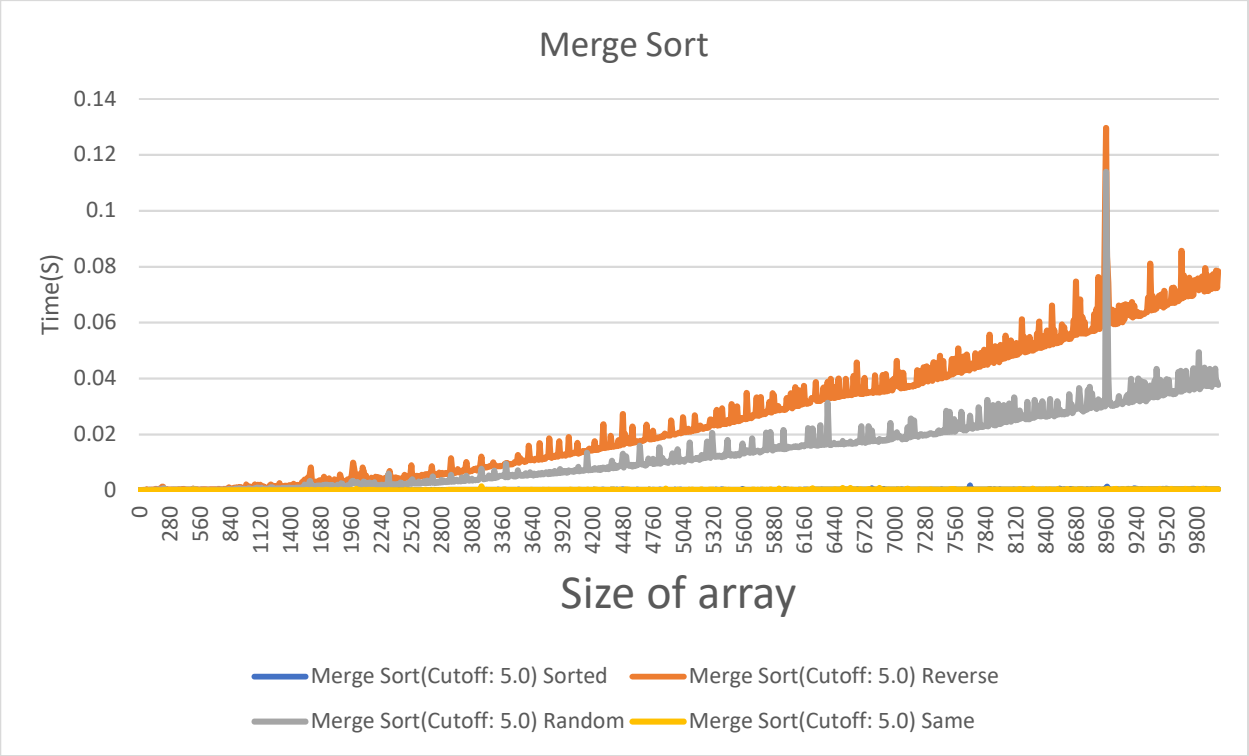
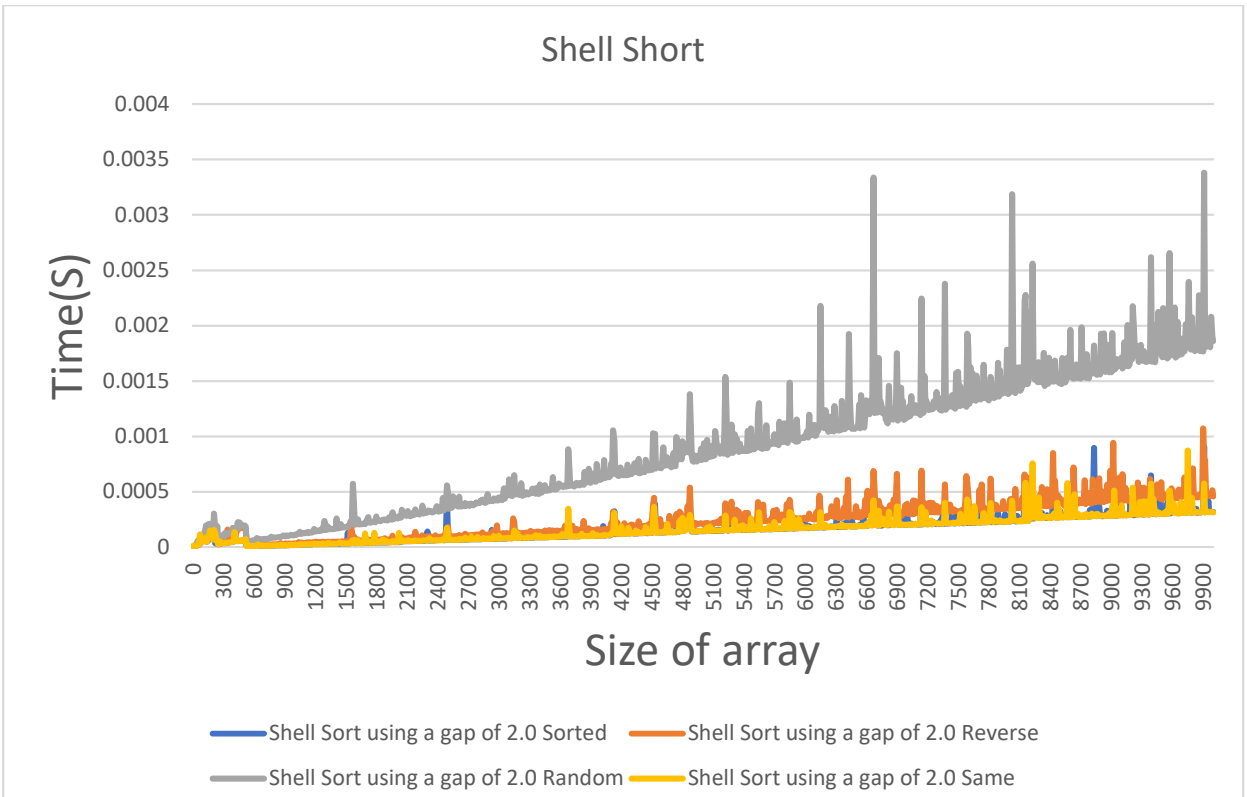
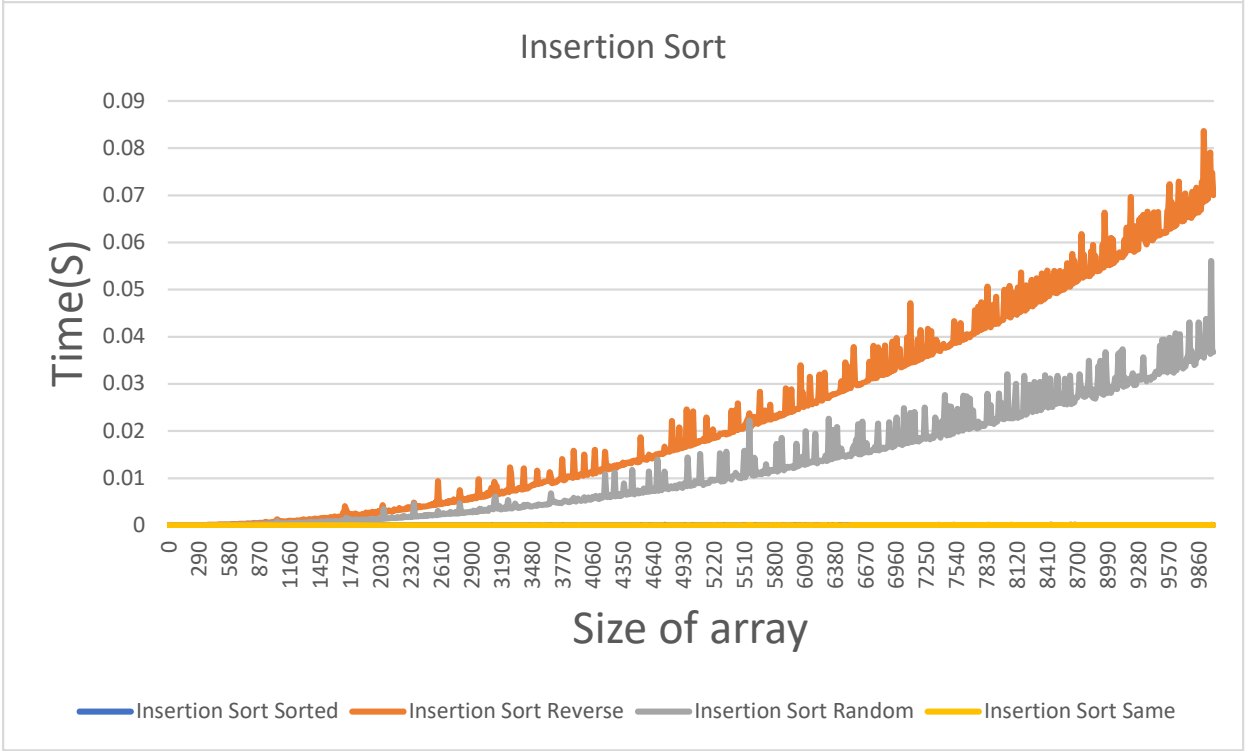
Nickolas Komarnitsky
02/16/2017
2420
Assignment 05

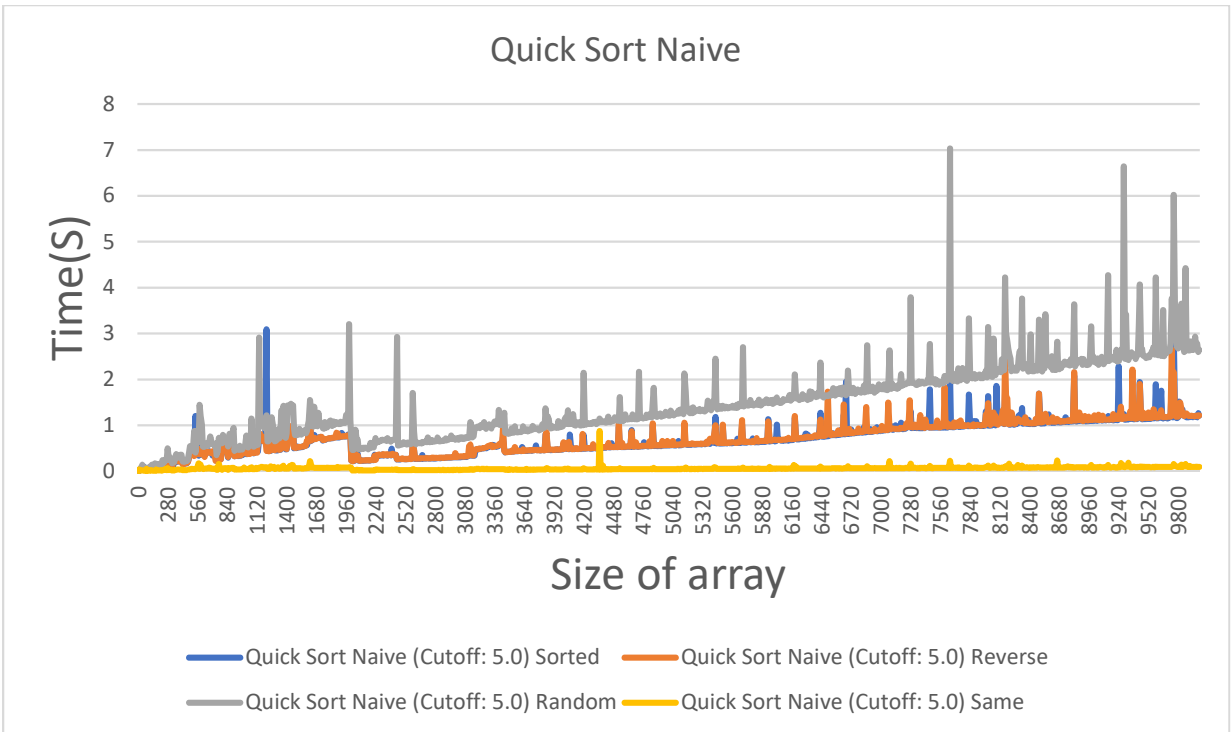
1. Graphs

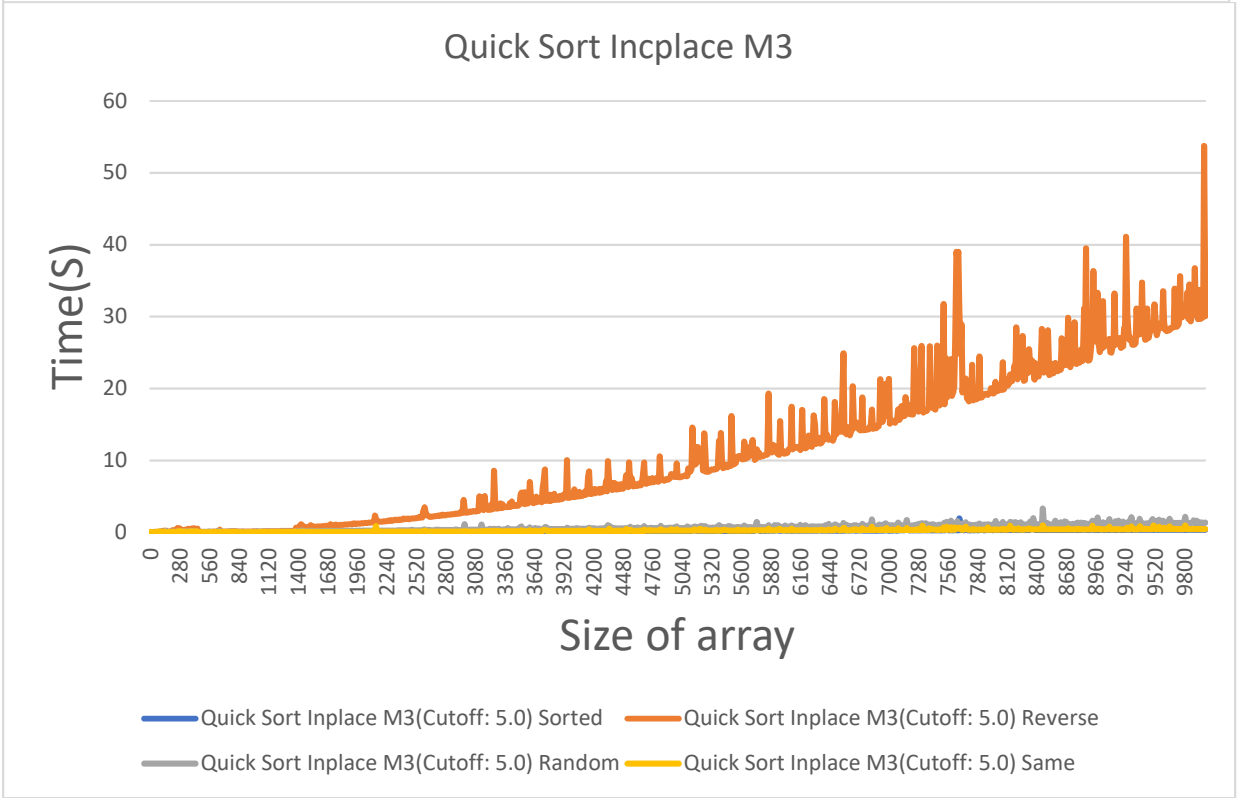


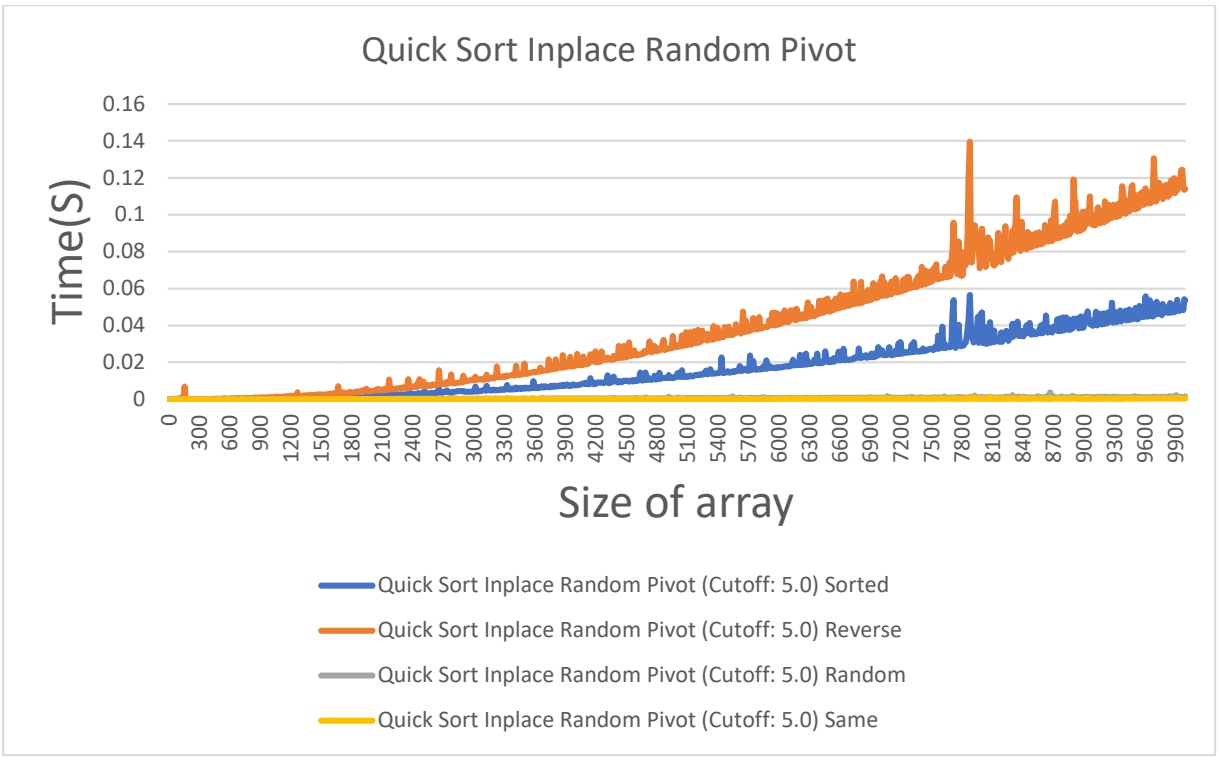


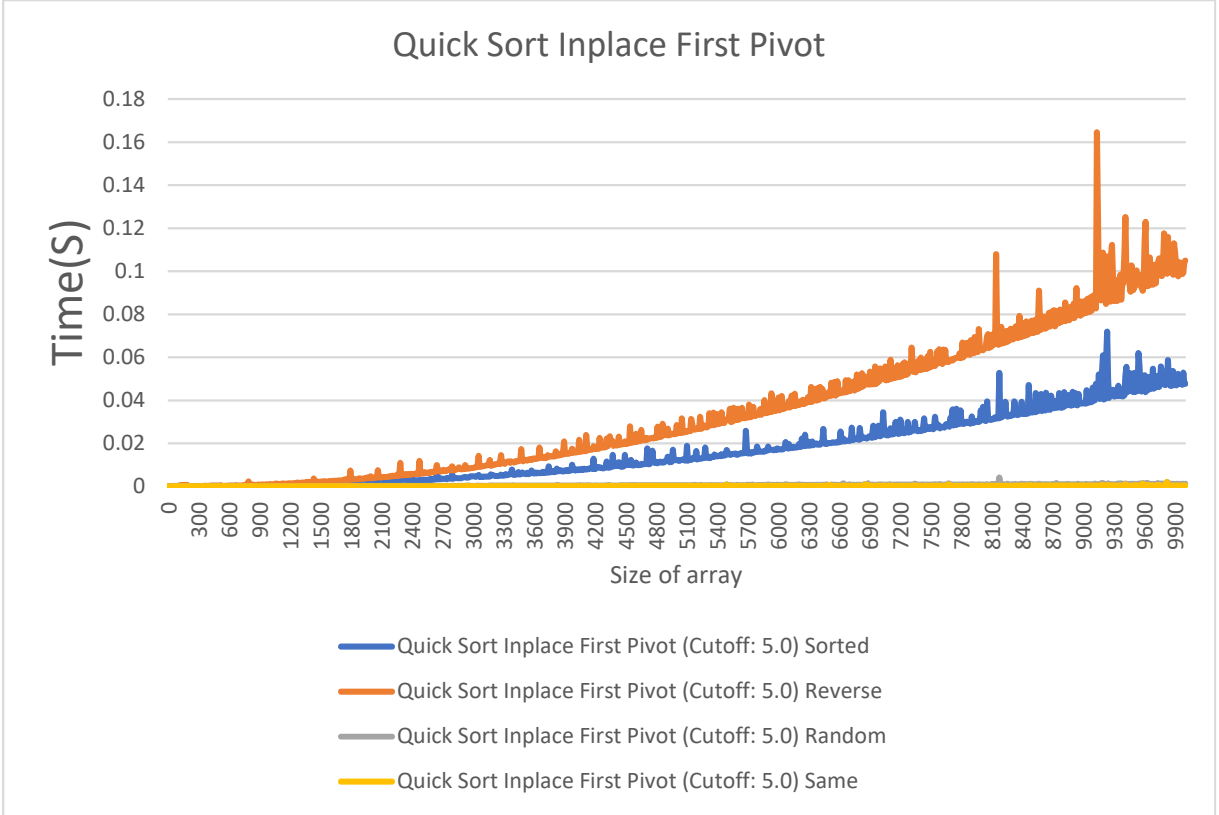

Shell Short
Time(S)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100 5400 5700 6000 6300 6600 6900 7200 7500 7800 8100 8400 8700 9000 9300 9600 9900
Size of array
Shell Sort using a gap of 2.0 Sorted
Shell Sort using a gap of 2.0 Reverse
Shell Sort using a gap of 2.0 Random
Shell Sort using a gap of 2.0 Same


Insertion Sort
Time(S)
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0 290 580 870 1160 1450 1740 2030 2320 2610 2900 3190 3480 3770 4060 4350 4640 4930 5220 5510 5800 6090 6380 6670 6960 7250 7540 7830 8120 8410 8700 8990 9280 9570 9860
Size of array
Insertion Sort Sorted
Insertion Sort Reverse
Insertion Sort Random
Insertion Sort Same


Quick Sort Naive
Time(S)
0
1
2
3
4
5
6
7
8
0 280 560 840 1120 1400 1680 1960 2240 2520 2800 3080 3360 3640 3920 4200 4480 4760 5040 5320 5600 5880 6160 6440 6720 7000 7280 7560 7840 8120 8400 8680 8960 9240 9520 9800
Size of array
Quick Sort Naive (Cutoff: 5.0) Sorted
Quick Sort Naive (Cutoff: 5.0) Reverse
Quick Sort Naive (Cutoff: 5.0) Random
Quick Sort Naive (Cutoff: 5.0) Same


Quick Sort Incplace M3
Time(S)
0
10
20
30
40
50
60
0 280 560 840 1120 1400 1680 1960 2240 2520 2800 3080 3360 3640 3920 4200 4480 4760 5040 5320 5600 5880 6160 6440 6720 7000 7280 7560 7840 8120 8400 8680 8960 9240 9520 9800
Size of array
Quick Sort Inplace M3(Cutoff: 5.0) Sorted
Quick Sort Inplace M3(Cutoff: 5.0) Reverse
Quick Sort Inplace M3(Cutoff: 5.0) Random
Quick Sort Inplace M3(Cutoff: 5.0) Same


Quick Sort Inplace Random Pivot
Time(S)
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100 5400 5700 6000 6300 6600 6900 7200 7500 7800 8100 8400 8700 9000 9300 9600 9900
Size of array
Quick Sort Inplace Random Pivot (Cutoff: 5.0) Sorted
Quick Sort Inplace Random Pivot (Cutoff: 5.0) Reverse
Quick Sort Inplace Random Pivot (Cutoff: 5.0) Random
Quick Sort Inplace Random Pivot (Cutoff: 5.0) Same


Quick Sort Inplace First Pivot
Time(S)
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100 5400 5700 6000 6300 6600 6900 7200 7500 7800 8100 8400 8700 9000 9300 9600 9900
Size of array
Quick Sort Inplace First Pivot (Cutoff: 5.0) Sorted
Quick Sort Inplace First Pivot (Cutoff: 5.0) Reverse
Quick Sort Inplace First Pivot (Cutoff: 5.0) Random
Quick Sort Inplace First Pivot (Cutoff: 5.0) Same

2. Answer this question: Quick sort works best when the partition is the middle element of the array which is (for normal data) very close to the average value of all elements in the array. Thus: A better pivot choice would seem to be to use the average value of the array. There are two flaws to this. Identify and discuss them.
   The average value could not exist in the array itself and then an extra value ends up being added to the array. Also, it could end up with values in the wrong places, comparing to a different average each time as a value is added to the array.

3. Answer this question: Why does the Fisher-Yates shuffle use the following line:
   j ← random interger from zero to i
   instead of
   j ← random interger from zero to N
   Because i is the maximum value that can be put into that spot, whereas N is the amount of items to add to the array.

4. Finally, discuss what you have learned about a) implementation tweaks to improve performance, and b) algorithmic choices to improve performance.
   I have learned that it can be fairly simple to figure out the algorithm and understand it, but then complex to make it work with actual code.