Connor Taylor
ASSGNJ

Factor = 1000

	Sort	stable sort	quicksort
1	0.000	0.000	0.000
2	0.001	0.001	0.000
4	0.001	0.001	0.001
8	0.002	0.002	0.001
16	0.006	0.004	0.002
32	0.008	0.008	0.008
64	0.024	0.020	0.013
128	0.045	0.040	0.030
256	0.093	0.088	0.062
512	0.189	0.183	0.126

Factor = 250

	Sort	stable sort	quicksort
1	0.000	0.000	0.000
2	0.000	0.000	0.000
4	0.000	0.001	0.000
8	0.000	0.001	0.001
16	0.002	0.001	0.001
32	0.000	0.005	0.001
64	0.005	0.005	0.005
128	0.012	0.011	0.007
256	0.024	0.020	0.012
512	0.044	0.040	0.029

To slow down my quicksort function, I altered the code by removing all functions except the main quicksort function and implemented the helper functions, swap and partition, inside the quicksort function. I believe the function in this implementation of the quicksort is slower because it uses the for loop from the partition function. Instead of it being destroyed from the stack after each call, it remains each time when quicksort is called.

	Sort	stable sort	quicksort
1	0.000	0.000	0.000
2	0.001	0.001	0.000
4	0.001	0.001	0.001
8	0.002	0.002	0.001
16	0.005	0.004	0.004
32	0.013	0.009	0.008
64	0.024	0.020	0.016
128	0.049	0.042	0.036
256	0.093	0.087	0.072
512	0.189	0.184	0.146

To attempt to speed up the quicksort function I made a small adjustment in the partition function and used the pre-increment operator instead post-increment operator. The change was very minimal, but it was still present. The reason for its improvement is that the post-increment operator copies the variable and then increments it whereas the pre-increment operator immediately increments the value, without taking a copy. Note that I had just tested the post-increment version again prior to immediately testing the pre-increment version and the post-increment version was 0.129.

	Sort	stable sort	quicksort
1	0.000	0.000	0.000
2	0.001	0.001	0.000
4	0.001	0.001	0.001
8	0.002	0.002	0.002
16	0.005	0.004	0.004
32	0.011	0.009	0.007
64	0.022	0.021	0.015
128	0.044	0.041	0.029
256	0.095	0.089	0.063
512	0.193	0.190	0.128