Algorithms and Data Structures
Reed College
Instructor: Erica Blum
Authors: Heath Nevis and Declan Bradley

Project 1: Assessing Sorting Runtimes

Compare your three algorithms (or four, if you implemented IntroSort) on a variety of test inputs. What conclusions can you draw?
The three sorting algorithms we analyzed were QuickSort, MergeSort, and SelectionSort. With a random array of varying size, Quicksort sorted it in roughly the same time regardless of the length of the array. MergeSort was about half as fast as QuickSort with the smallest array, but got even slower as the size of the array increased. SelectionSort was by far the slowest, starting off at almost 60 times slower than QuickSort, and growing significantly slower as the array's size increased. From this data, we can see that of the three sorting algorithms, QuickSort is the fastest, followed by MergeSort and finally SelectionSort. Because we used random arrays, this conclusion is only relevant when working with random arrays. The relative speeds of the algorithms are different when the array being sorted is already sorted or in 'evil' order.

python sortTest.py 1000 500 2000 100 -g random

Size of Array	QuickSort	MergeSort	SelectionSort
1,000	0.00101754	0.00281974	0.06416116
1,500	0.00210864	0.00500383	0.15666925
2,000	0.00149668	0.00675217	0.49866055

Collect and present evidence that your evil generator works, i.e., that QuickSort runs more slowly on your evil inputs than random ones.
When the array is in evil mode, it is arranged to be as bad as possible for QuickSort, which happens when the array is in reverse order. This is shown by comparing the runtime of QuickSort with a random array versus an evil array. For all the sizes of arrays listed here, QuickSort ran over an order of magnitude slower with an evil array. Depending on the size of the array, QuickSort is between 28 and 73 times slower on an evil array than on a random array. From this comparison, it is clear that QuickSort runs significantly slower on a reverse sorted (evil) array than on a random one.

python sortTest.py 1000 500 2000 100 -g evil

Size of Array	QuickSort	MergeSort	SelectionSort
1,000	0.02845507	0.00242267	0.06200423
1,500	0.06356528	0.00408156	0.1530371
2,000	0.11010873	0.00619266	0.28131599