Brennan McFarland
April 23, 2016
Programming Assignment #4 Report

To compare the performance of heapSort, quickSort and mergeSort, I ran each algorithm on differently sized arrays with both sorted input, reverse sorted input and random input. For the sorted and reverse sorted input, I ran each specific scenario for each algorithm three times and took the median to reduce chance variances, and for the randomly sorted input I ran each scenario three times and took the mean. Each sorting method recorded its running time in nanoseconds, and the reporting program wrote these running times to a text file. The formatted results, as well as the variances for randomly sorted input, are shown in the three tables below:

Sorted Input (median of 3):

Array Size:	1,000	10,000	100,000	1,000,000
HeapSort:	424,034	2,415,532	8,619,201	115,832,794
QuickSort:	1,259,344	3,142,933	2,763,760	34,260,842
MergeSort:	1,158,570	301,946	5,019,850	58,612,672

Reverse Sorted Input (median of 3):

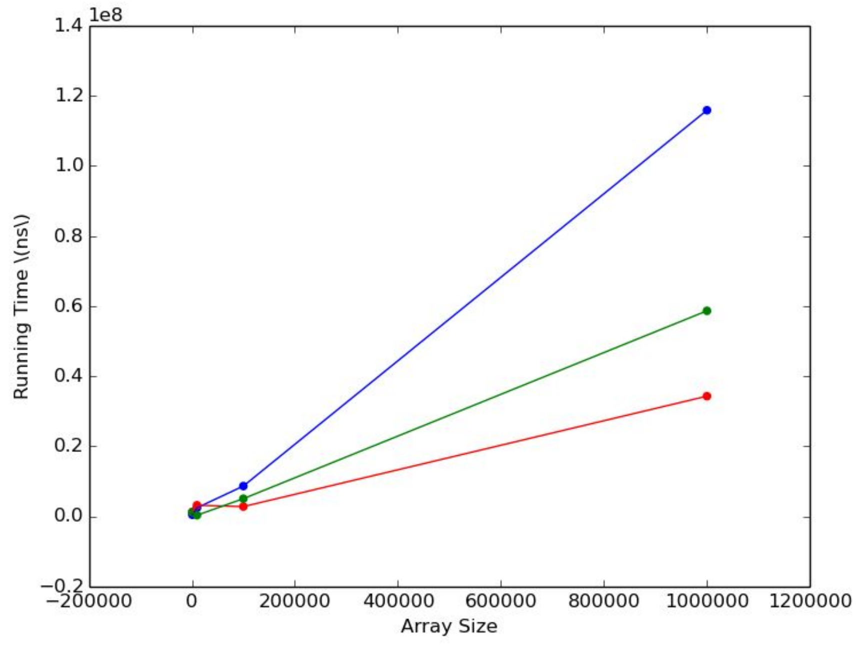
Array Size:	1,000	10,000	100,000	1,000,000
HeapSort:	57,267	680,848	8,660,677	114,811,565
QuickSort:	185,240	2,280,115	2,871,455	33,899,842
MergeSort:	924,655	704,638	5,402,852	63,148,096

Random Input(mean of 3):

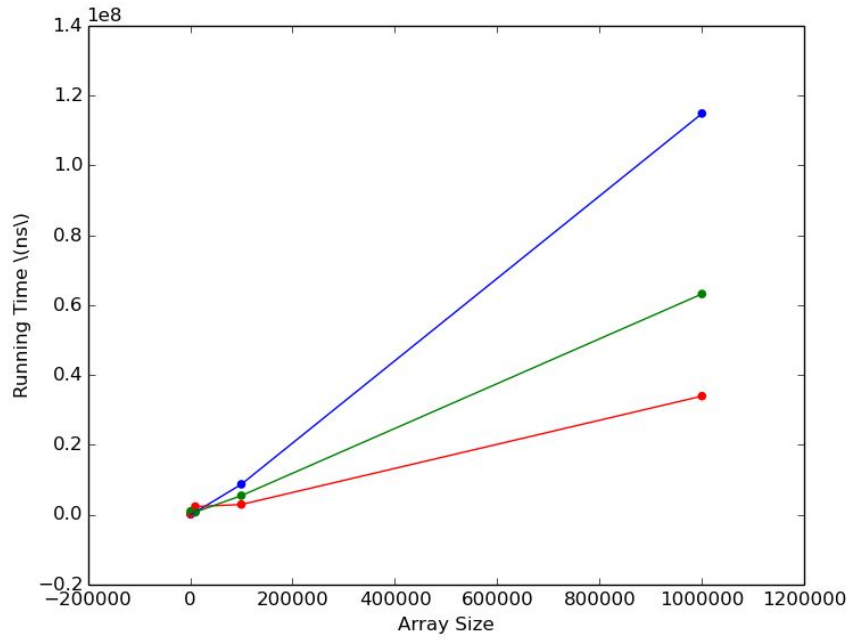
Array Size:	1,000	10,000	100,000	1,000,000
HeapSort:	74,582	967,071	12,868,577	226,602,059
QuickSort:	83,209	1,015,219	12,748,436	151,498,923
MergeSort:	73,582	1,060,676	13,428,428	165,626,575
Variance:	28,017,377	2,191,077,463	131,709,353,700	1,592,559,994,000,000

Graphing these results, it becomes clear that quickSort is generally the fastest algorithm, leaving mergeSort in close second and heapSort in last, taking the longest time to sort array contents. The graphical representation of these results is given below, with red representing quickSort, green representing mergeSort and blue representing heapSort:

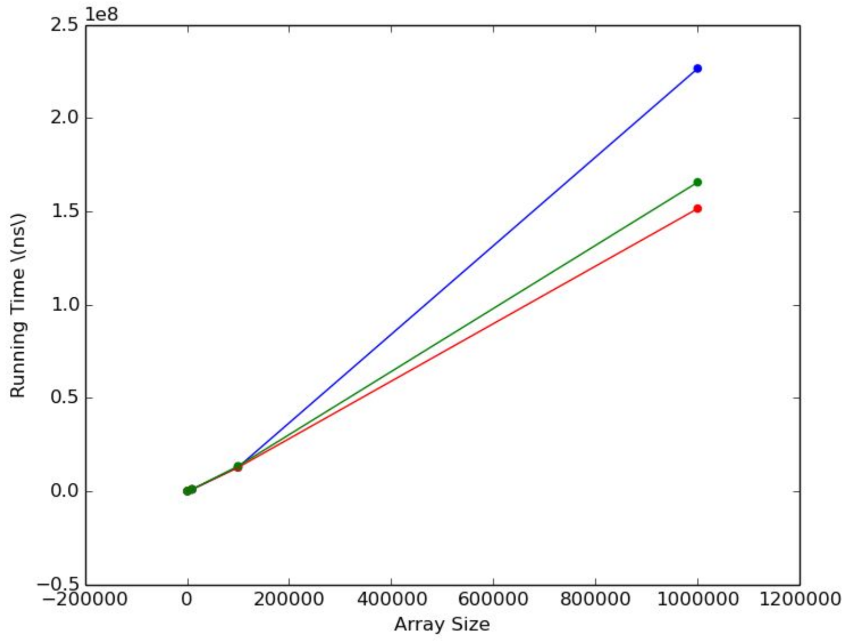
Sorted Input (median of 3):



Reverse Sorted Input (median of 3):



Random Input (mean of 3):



The dots represent the actual trials, whereas the lines interpolate linearly between them in order to distinguish each algorithm more clearly. Thus the ranking, from most efficient to least efficient, of the three algorithms in every case is as follows: quickSort, mergeSort, heapSort. Thus is in agreement with the results of the other reporting program designed to test the sorting implementation:

HS bfm21 = 20805ns; QS bfm21 = 10728ns; MS bfm21 = 48261ns;

In terms of the type of input data, heapSort falls behind quickSort and mergeSort the most (proportionally) with randomized data, and quickSort and mergeSort become less than distinguishable from each other. As indicated by the graphs, runtime is almost the same between the algorithms for small arrays, with quickSort taking slightly longer and the difference between heapSort and mergeSort insignificant, and as expected the differences in efficiency become greater as the input array size increases. Additionally, the variance of the random sorting increases with array size as expected.

Thus, it appears evident that in every case, quickSort is the most efficient algorithm. It is known that quickSort can be less efficient in certain cases depending on pivot selection, but these rare cases either did not occur in the experimental data or were averaged out in the multiple trials. Therefore, for smaller arrays the data indicates that either heapSort or mergeSort should be used, whereas for arrays of significant size, say, above 200,000 elements for an integer array, quickSort is by a good margin the most efficient algorithm.