David Fan
Dmf98

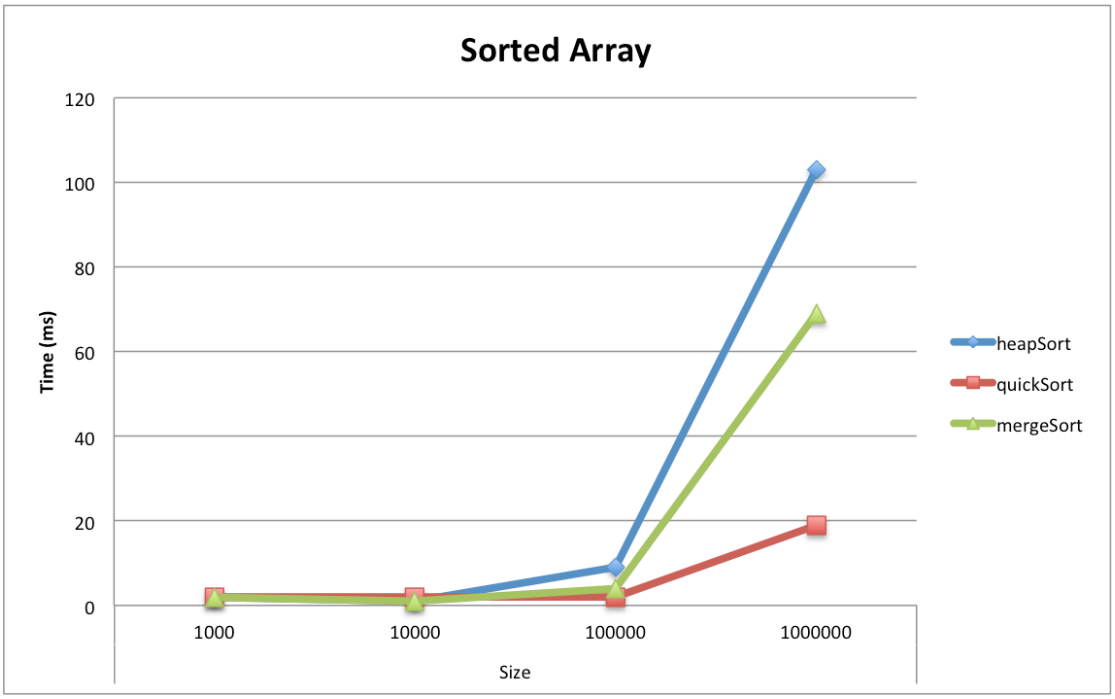
Sorting Algorithm Report

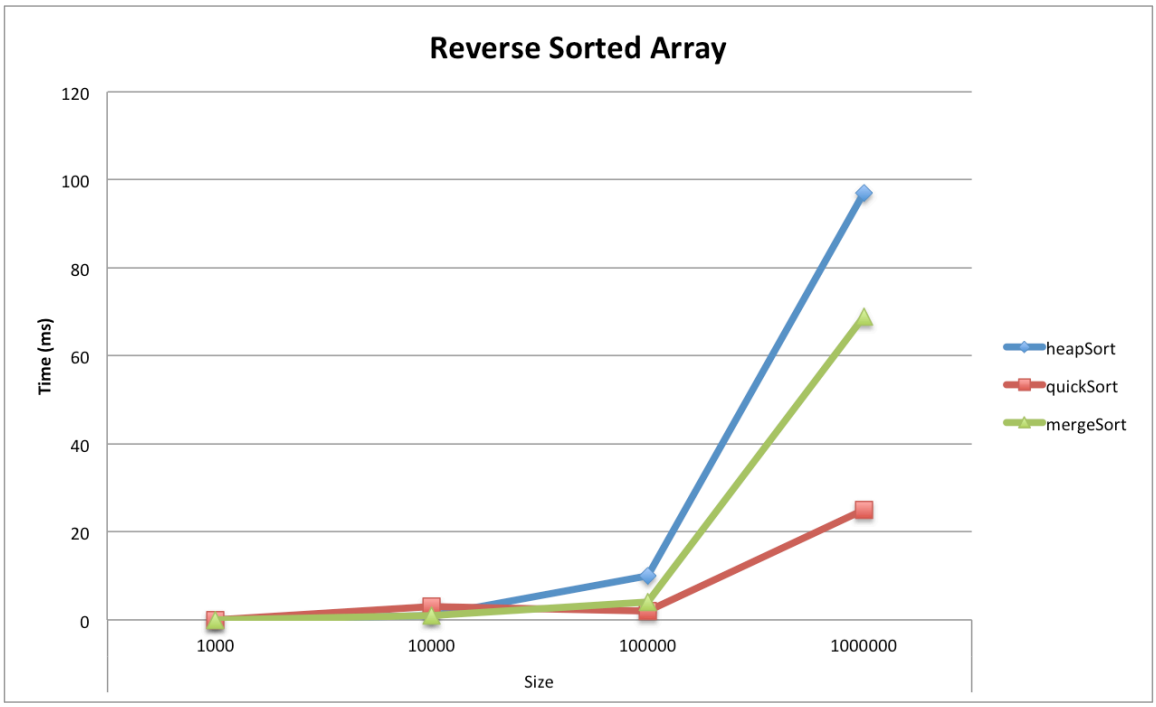
Conclusions from data: The data indicates that quick sort is the superior algorithm to heap sort and merge sort, as it took significantly less time than the other two algorithms to execute at larger input array sizes. This conclusion holds across the sorted array, the reverse-sorted array, and the randomly generated array.

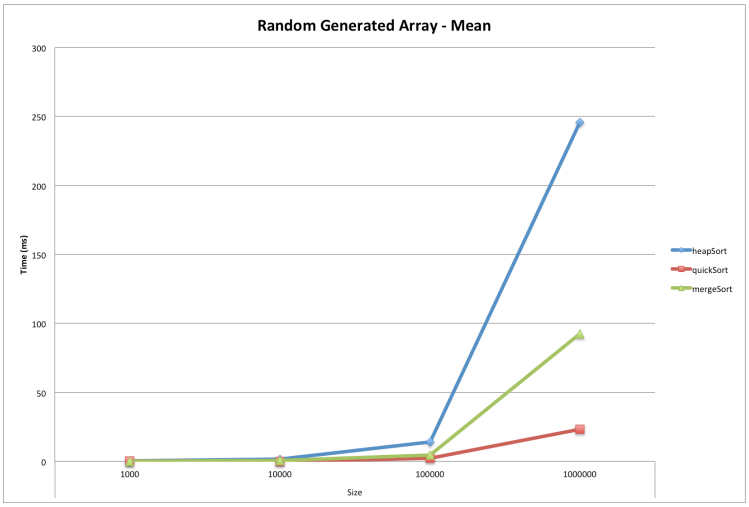
Interpretations: The results are questionable as quick sort theoretically has a greater worst-case run time at $O(n^2)$ while both merge sort and heap sort have a worst-case run time of $O(n \log(n))$. This might indicate that my implementation of quick sort was more efficient than my implementations of heap sort and merge sort.

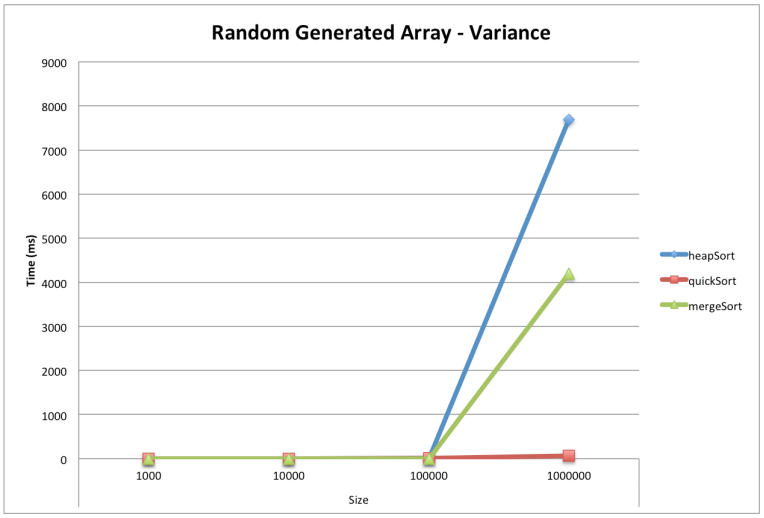
Results from Reporting1.java class: I outputted the results from my Reporting1.java class to a file in a format that allowed for post-processing in excel. However, to maintain uniformity in my output (to not have many more results for the random array than the sorted and reverse-sorted arrays) I calculated the mean and variance in my Reporting1.java class using the required helper methods. My output file formatted the results into columns separated by spaces allowing me to easily create charts from the data.

Charts:









Tables:

Sorted Array – Median Execution Times			
Size	heapSort	quickSort	mergeSort
1000	2	2	2
10000	1	2	1
100000	9	2	4
1000000	103	19	69

Reverse Sorted Array – Median Execution Times			
Size	heapSort	quickSort	mergeSort
1000	0	0	0
10000	1	3	1
100000	10	2	4
1000000	97	25	69

Randomly Generated Array - Mean			
Size	heapSort	quickSort	mergeSort
1000	0.4	0.4	0.1
10000	1.7	0	0.7
100000	14.2	2.4	4.7
1000000	246	23.2	92.4

Randomly Generated Array - Variance			
Size	heapSort	quickSort	mergeSort
1000	0.266666667	0.266666667	0.1
10000	1.788888889	0	1.566666667
100000	7.955555556	4.266666667	1.788888889
1000000	7693.555556	63.95555556	4195.155556