**Algorithm Combination Proposal:**

An algorithm I want to propose is the quick and insertion sort. The reasoning behind this combination since quick sort does a good job in sorting lists quickly and making them nearly sorted midway the algorithm. Insertion sort can come in and sort the list more easily due quick sort making the array nearly or somewhat sorted. It should have an overall good time complexity.

***Pseudocode of proposed algorithm (quick/sort):***

**void (int[] array, int lowvalue, int highvalue)**

if lowvalue < highvalue

int pivot = quickSortpartitonMethod(array, lowvalue, highvalue)

//calls upon quicksortpartition method to get pivot

quickSort(arrlength, lowvalue, pivot - 1)

quickSort(arrlength, pivot + 1, highvalue)

integer: arraylength = arrayofthelength

for loop: i=1 ; i < arraylength; i=i+1

int keyvalue = array[i]

int j = i - 1

while loop: (j >= 0) AND (array[j] > keyvalue)

array[j + 1] = array[j]

j = j - 1

array[j + 1] = keyvalue

**Testing Strategy Description:**

Measuring the performance of the algorithms is important to find which sorting algorithms perform best in different (or most) scenarios. For this, I would be using time to measure the performance of the algorithms. One will be a counter that can measure the steps in each algorithm. The second one is to measure the time (in milliseconds outputted from the algorithm). These algorithms will be tested with different kinds of arrays (some suppose to be worst/best case scenarios for that algorithm). The rest will be gathering data and comparing it among the different kind of sorting algorithms.

The five test cases required for this lab will be five different arrays. These arrays will be sorted (or unsorted) differently, and with different lengths.

Array 1: {6, 2, 3, 4, 23, 5, 54, 69, 69, 231, 32, 0, 8, 7, 1, 9, 23, 45, 62}

Array 2: {1, 3, 2}

Array 3: {70, 65, 50, 1, 40, 41, 71, 84}

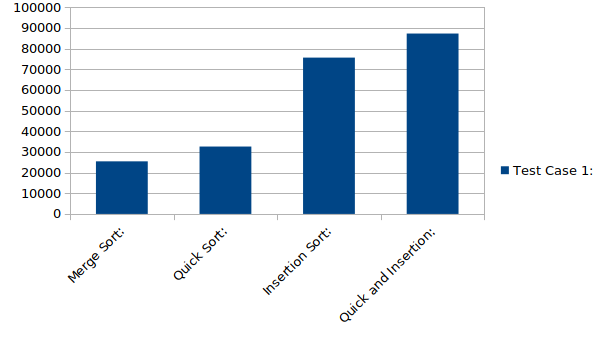
Array 4: {1, 1, 3, 8, 8, 4, 1}

Array 5: {9, 7, 6, 5, 4, 3, 2, 1}

**Results of Testing:**

*Summary:*  After a series of testing each sorting method in all five test cases, merge sort and quick came out to be the best sorting methods with lower times than insertion and quick/insertion. My proposed sorting method (a combination of quick and insertion sort) did not yield better results than quick sort, merge, and insertion sort. If anything, lagged behind the other sorting algorithms and did not offer any performance gain. Insertion sort was comparable to quick/insertion sort (my proposed algorithm).

Test Case 1 (Array 1):

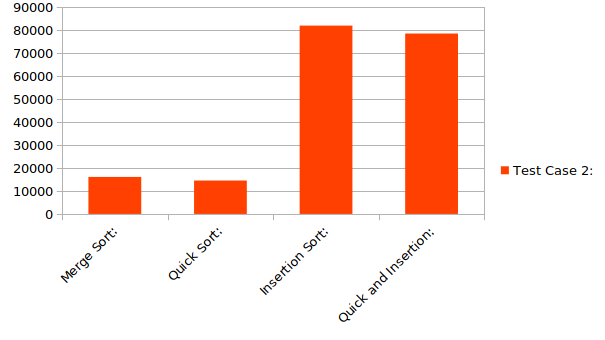
 Merge: 25460

Quick : 32601

Insertion: 75599

Quick/Insertion: 87265

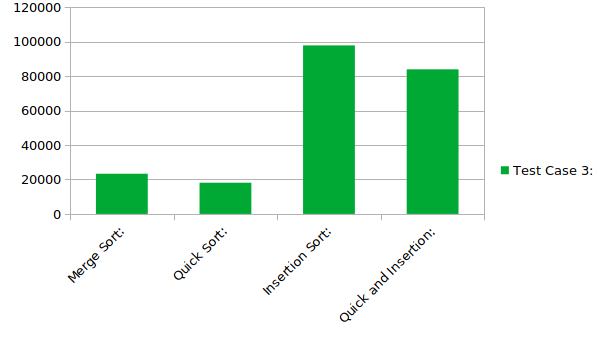
Test Case 2 (Array 2):

 Merge: 16118

Quick: 14528

Insertion: 81996

Quick/Insertion: 78544

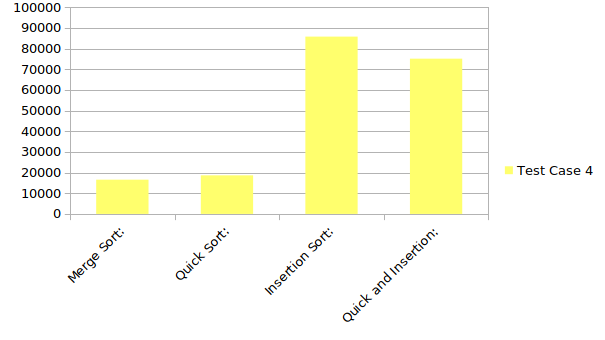
 Test Case 3 (Array 3):

Merge: 23326

Quick: 18107

Insertion: 97854

Quick/Insertion 83947

 Test Case 4 (Array 4):

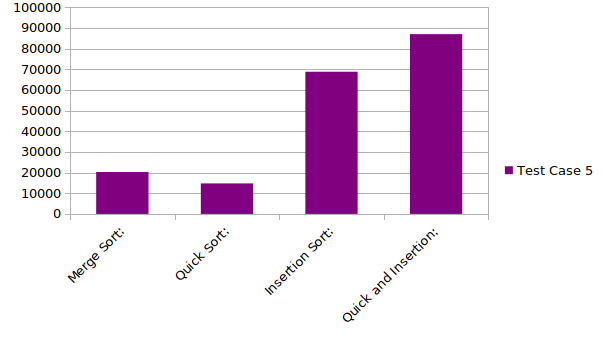
Merge:

Quick:

Insertion:

Quick/Insertion:

Test Case 5 (Array 5):

 Merge:

Quick:

Insertion:

Quick/Insertion:

**Conclusions:**

My hypothesis was that quick/insertion sort was the best combination of a sorting algorithm. But after testing all the algorithms, my combination of quick/insertion sort was not the best contender among the sorting algorithms. It consistently lagged behind merge sort, and was comparable to quick sort and insertion sort in terms of times. While it performed better in some flavors of arrays, it couldn’t beat merge sort in all testing scenarios.

My proposed combination of algorithms didn’t perform as well I expected it to, so my initial hypothesis was wrong. I believed that insertion sort would have performed better in a partially sorted array, thus reducing time, but the bottlenecks in that algorithm cannot be curb by partially sorting the array.

**Follow Up Sorting Projects:**

A follow up project that I also want to propose is a quick/bubble sort combination. During the testing of the different sorting algorithms, I realized the bubble sort would have probably done better in reducing in time complexity than insertion sort. There are less “bottlenecks” in bubble sort, and it would have probably benefited better from a partial sorted array than insertion sort.