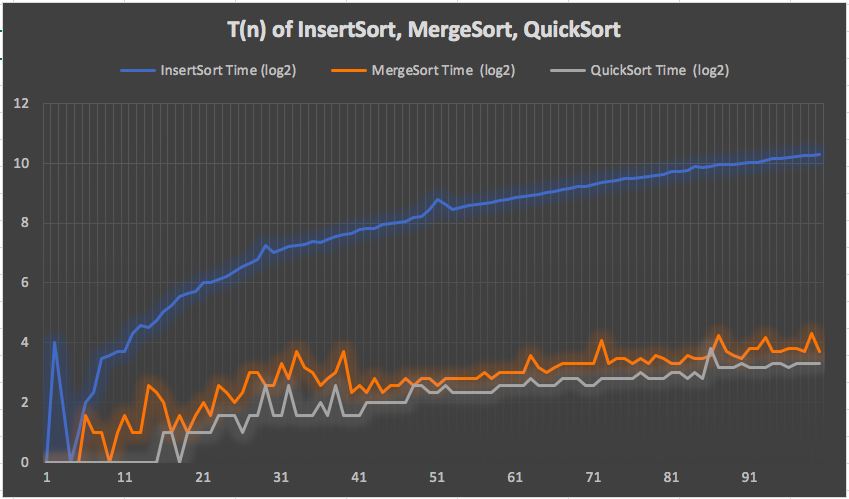
Greg Maxwell  
Zamir Thind  
  
CPSC 3273 Programming Assignment 2

* Does the code work? Yes
* Compilation/execution: Simply compile the source file(s) within a Java IDE.
* Analysis of graph: Our graph starts at n = 10. The running times of algorithms Insert Sort, Merge Sort and Quick Sort are plotted in increments increasing by 1000 for each iteration. We noticed that Insert Sort peaked around n = 1010, then dipped until it overtook Merge Sort and Quick Sort around n = 7010. We believe this is an anomaly and not representative of the algorithm's typical performance.  
    
  As the graph states, we can determine that Insert Sort takes a longer time by far to sort larger input sizes compared to the two other algorithms. Insert Sort shows an average growth rate of n2, which is in line with expectations. Merge Sort and Quick Sort, on the other hand, show much slower and controlled growth rates. These two algorithms show an average growth rate of n lg n. Interestingly enough, despite having the same average time complexity, Quick Sort was a bit faster overall. The worst case time complexity for Quick Sort is n2, unlike Merge Sort, which has a worst case time complexity of n lg n.
* Conclusion: Merge Sort and Quick Sort are far more efficient sorting algorithms compared to Insert Sort, which takes a considerable time longer to sort large input sizes. Curiously, we noticed that Quick Sort is a little bit faster on average than Merge Sort. Our hypothesis is that, as Merge Sort is not an in-place algorithm, the act of creating and merging the sorted arrays increases the time complexity by a small, but not insignificant, amount.