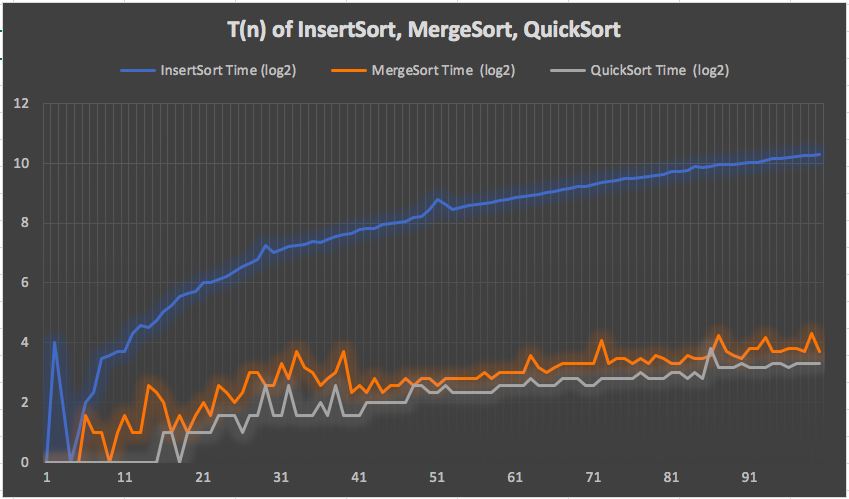
Greg Maxwell  
Zamir Thind  
  
CPSC 3273 Programming Assignment 3

* Does the code work? Yes
* Compilation/execution: Simply compile the source file(s) within a Java IDE.
* Analysis of graph: Our objective is to measure the running time of three different sorting algorithms: Insert Sort, Quick Sort and Merge Sort. The running time of T(n) is measured for each algorithm during each iteration of the sorting process. Our graph starts at n = 10. The amount of elements to sort increases by 1000 for each iteration until n = 100000. In the beginning of our program, we noticed that Insert Sort spiked around n = 1010, then came back down before overtaking the other algorithms at n = 7010. We believe this is an anomaly and not representative of the algorithm's typical performance.  
    
   Based off of the chart, we can see that Insert Sort resembles a growth rate of n2, as the running time dramatically increases as input size grows. Merge Sort and Quick Sort exhibit a similar growth rate of n lg n, with running time growing in a slow, controlled manner. The most interesting thing about these two algorithms is that Quick Sort is growing at a slightly slower rate than Merge Sort, despite the worst case scenario for Quick Sort having a time complexity of n2 . For comparison's sake, the worst case scenario for Merge Sort holds a time complexity of n lg n.
* Conclusion: Merge Sort and Quick Sort are orders of magnitude more efficient than the sluggish Insert Sort, which quickly grows in time complexity in line with input size. Interestingly enough, Quick Sort does seem to hold a bit of an advantage in running time over Merge Sort, despite a more timely worst case scenario of n2. Our hypothesis is that the act of creating and merging the sorted arrays during Merge Sort contributes to a slightly higher running time.