Scott Mueller
ComS 311
Hmwk 4

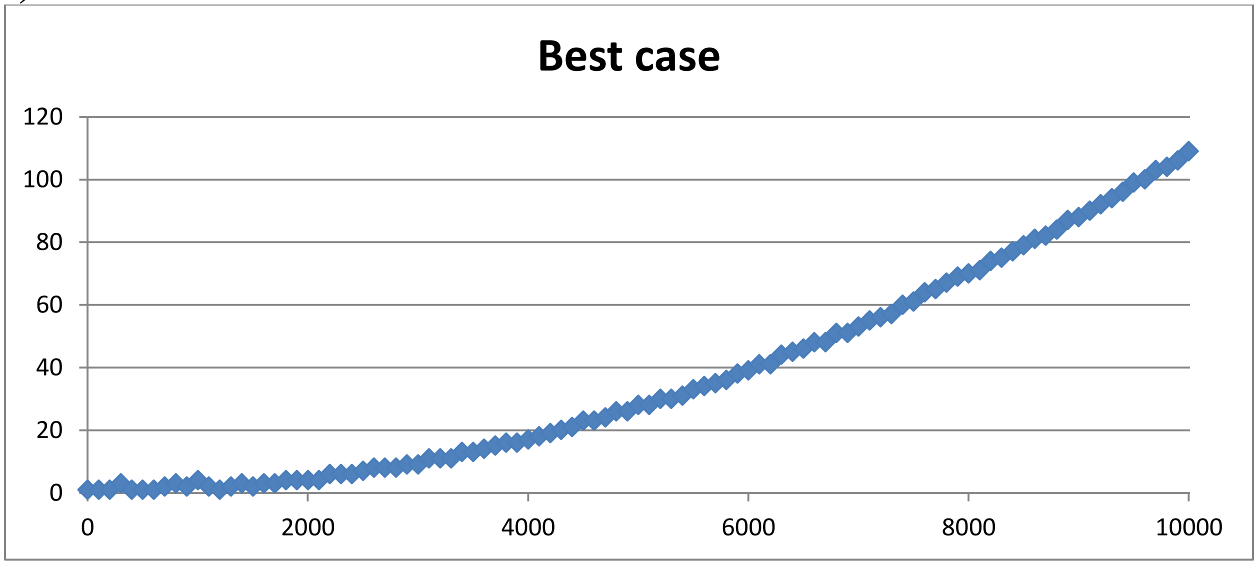
Part 2:
a) The “best case” array for my algorithm would be an array of size n in which the elements between the given indexes are already sorted correctly.

b) The “Big Omega” lower bound of the best case running time of the algorithm is $\Omega(n)$. This is due to the fact that even if the array is sorted, the algorithm will still need to run through it and check every element to make sure it is greater than the one before it and less than the one after it.

c) The “worst case” array for my algorithm would be an array of size n in which the elements between the given indexes are sorted in reverse order. For instance, all elements are sorted in a descending order with the larger elements at lower indexes and the smaller elements at higher indexes when we want the array sorted in the opposite way.

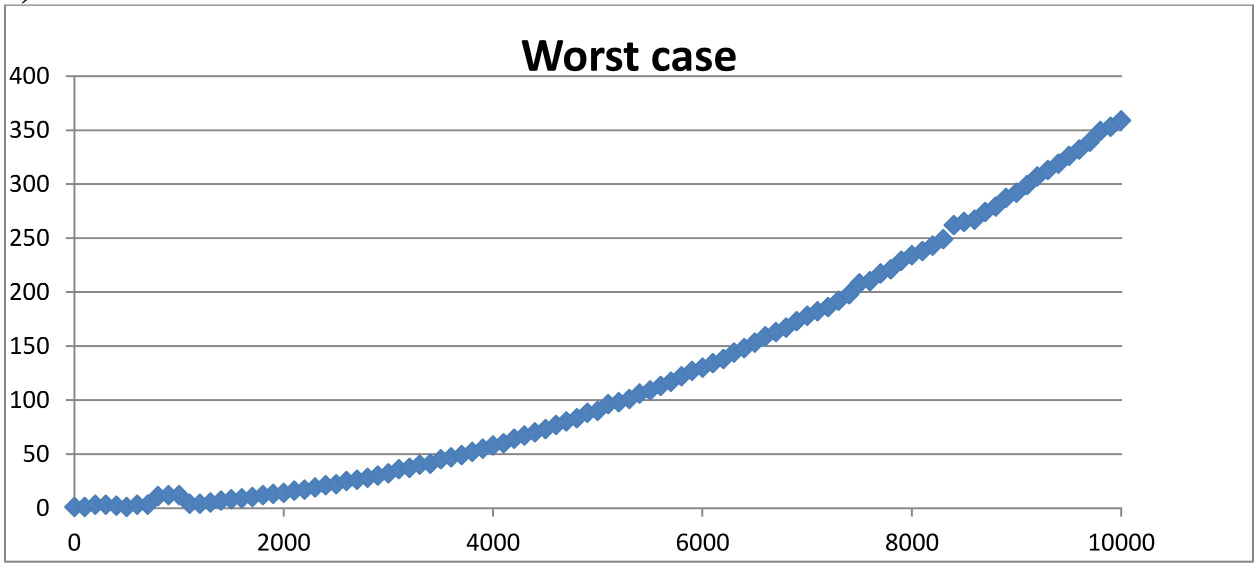
d) The “Big O” upper bound for the worst case running time of the algorithm is $O(n^2)$. This is due to the fact that it will need to go through and swap every element at a high index with the ones at lower indexes than it and will ultimately need to run through the array making n^2 comparisons.

Part3:
a)



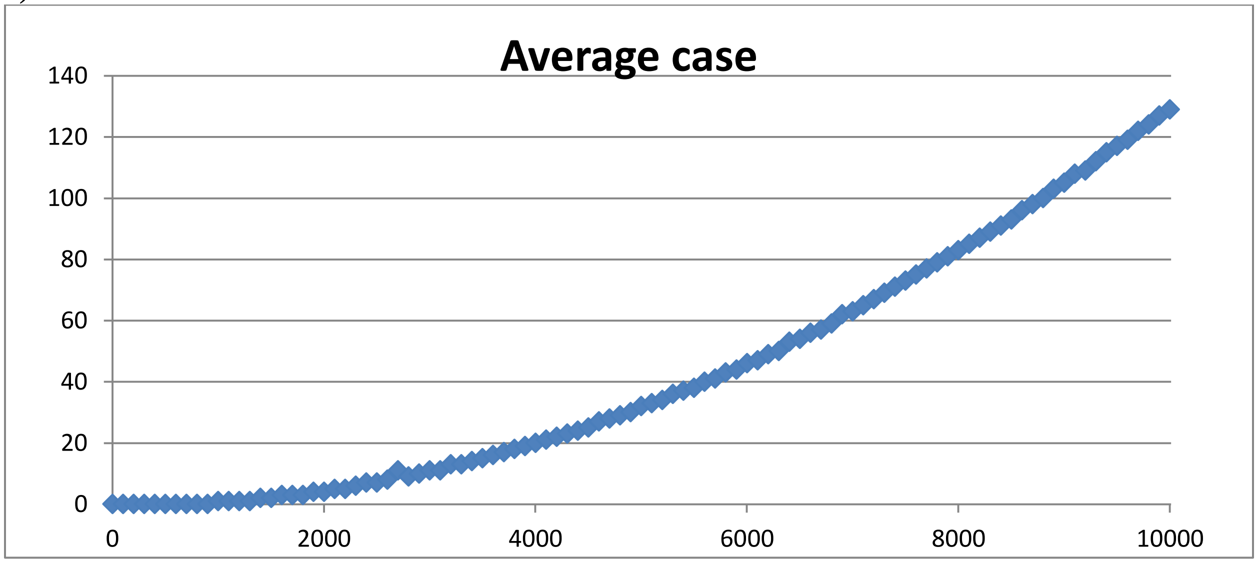
Our scatter plot as we can see almost looks rather linear. For arrays below size n=3000 the algorithm runs quickly and it is hard get good data. However, if we look at arrays of size n>=3000, we can see that the points increase at a constant, linear rate.

b)



Our scatter plot in the worst case scenario takes on a quadratic appearance. As we can see here it curves to start the formation of a half U shape as expected for something that runs in n^2 running time.

c)



Our scatter plot for an average case run of our algorithm take on an appearance similar to our worst case. This does make sense since an average case run of an insertion sort algorithm is going to be making close to n^2 comparisons and thus running in n^2 time.