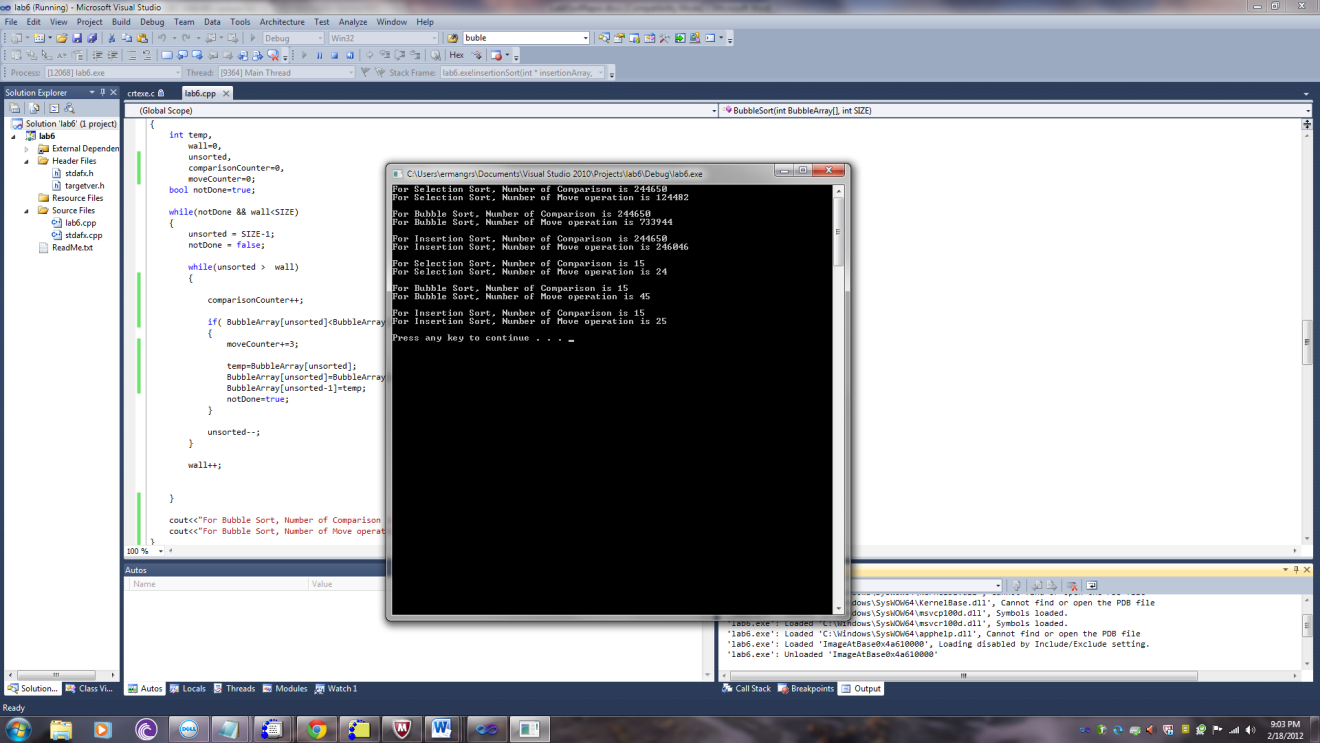
COSC 1436 – Intro to Problem Solving II

Lab 6 Sort Report

# Erman Gurses

**February 22, 2012**



Yes I agree with Big-O efficiencies for all three sorting algorithms is O(*f*(*n*)) = O(*n*2) because numbers of comparisons are same for three of them as we can see above. Number of comparison operations depends on *n* and *sorting sequence* but just one comparison operation takes constant time so big oh notation supports my data. By the way, there are different numbers of move operation but it does not affect my algorithmic complexity because move operations take constant time.

They have same number of comparison but different number of move operations. Bubble sort has the greatest number of move operation. It depends on type of comparison and type of move operation to determine a move or a comparison takes more time. If you compare elements of array which contain complex indices for example unsorted-1-I, it will take more time than simple move operation or vice versa. Also it depends on instruction set of CPU, maybe one CPU does comparison in a one clock cycle another one does 3.

If we talk about, which sorting algorithm is the most efficient for a set of 700 integers that are in reverse order? The algorithmic efficiency is same for three of them because numbers of comparisons are same and the algorithmic complexity is *O(n2)*

If we think random order we should think average case which means all permutations of input sequence are equally likely do it does not affect efficiency. It is still *O(n2)*

If the data are in ascending order, which means that the best case, the number of comparisons and the number of moves will not occur so much. So they will be fast and does not behave something, which ordered n2 they will behave as linear.

These 3 sorting algorithms can be overhead to have to keep the data ordered in order to perform a binary search because algorithmic complexity of three is O(n2) but algorithmic complexity of binary search is O(logn) so n2 is bigger term than logn so n2 is overhead for binary search. I can choice binary search in some situation even if I have to do sort for example, we can implement another kind of sorting algorithm for instance merge sort which has complexity nlogn or radix sort O(n+k) so it can be lower overhead. Maybe in some situation, we need to implement sorting one time but need to implement searching many times so we can still use binary search with these sorting algorithms.