Daniel Weitman

COSC-320

(a)

n = Array Size

Bubble Sort

Best Case: Omega(n)

Worst Case: O(n^2)

Insertion Sort

Best Case: Omega(n)

Worst Case: O(n^2)

Selection Sort

Best Case: Omega(n^2)

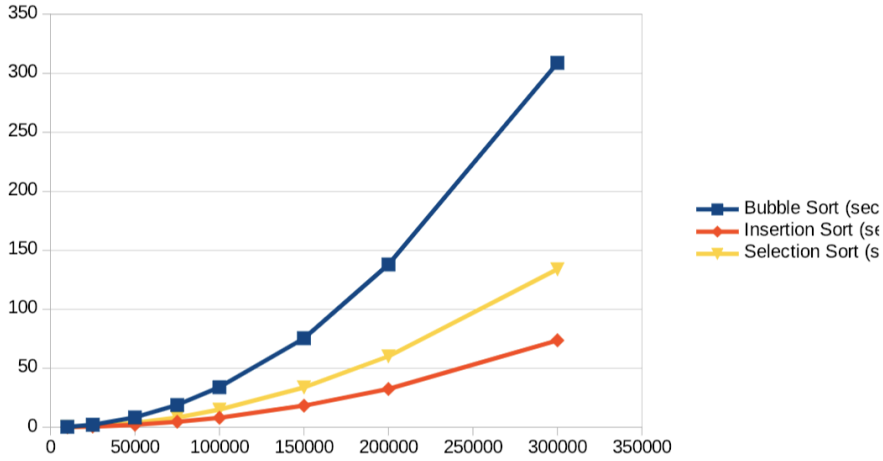
Worst Case: O(n^2)

(b)

The absolute timing complexity scales exponentially with number of element and/or element sizes. The absolute timing complexity lines up with the theoretical timing complexity.

(c)

Array size	Bubble Sort (sec)	Insertion Sort (sec)	Selection Sort (sec)
10000	0.397599	0.128842	0.217218
25000	2.14641	0.655842	1.07725
50000	8.42549	2.15844	3.78088
75000	18.9153	4.616	8.331333
100000	33.9987	8.00875	14.9553
150000	75.4479	18.2819	33.7789
200000	137.917	32.4327	60.1002
300000	308.772	73.7053	133.888



(d)

The insertion sort consistently sorted the array faster for all array sizes tested, whereas the bubble sort consistently sorted the slowest and the selection sort in between the them. Therefore, no algorithms were better at sorting smaller arrays then large arrays and vice versa, for times tested. For the smaller array sizes (< 10,000) the time discrepancy between all three algorithms was negligible, as they were all completed in under a second. However, as array sizes increased the discrepancy between the sorting times grew as well. All three algorithms displayed and exponential trend in their respective graph.

(e)

Other than adding more efficient sorting algorithms the code is executed well. Additionally, adding some more input validation may be useful in some instances.