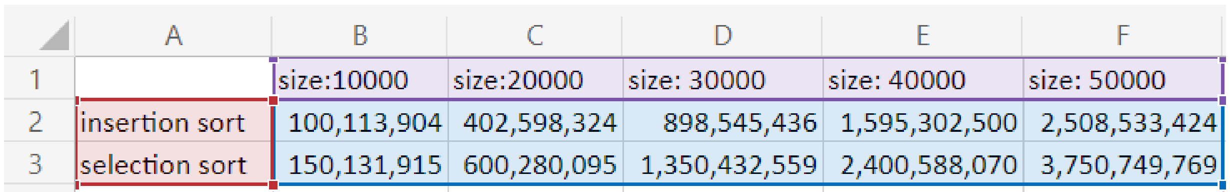
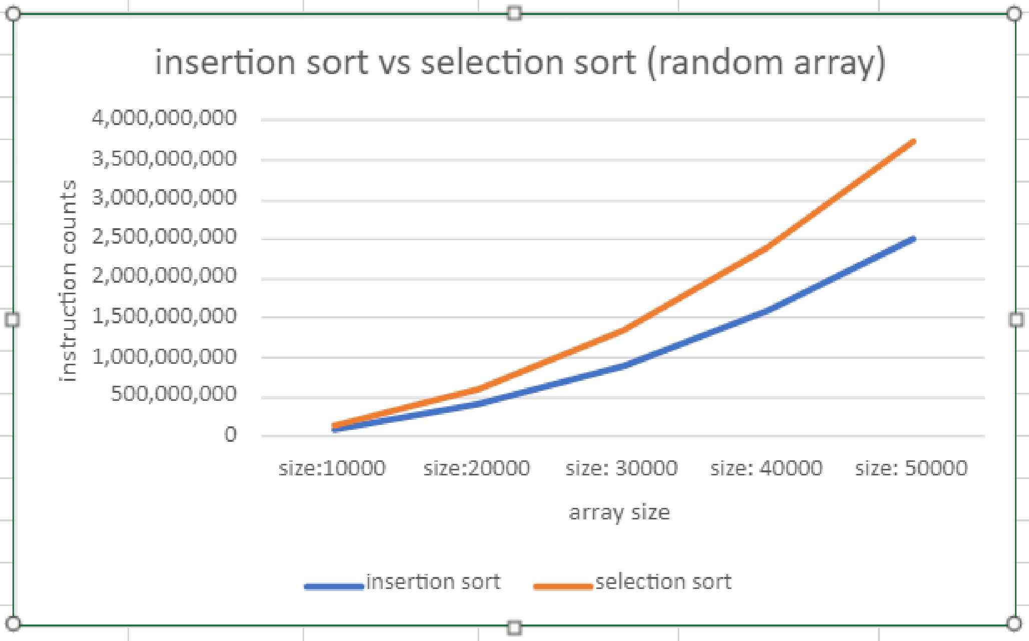


	A	B	C	D	E	F
1		size:10000	size:20000	size: 30000	size: 40000	size: 50000
2	insertion sort	100,113,904	402,598,324	898,545,436	1,595,302,500	2,508,533,424
3	selection sort	150,131,915	600,280,095	1,350,432,559	2,400,588,070	3,750,749,769



As can be seen from the data and graph, the insertion sort algorithm takes less time to execute than the selection sort algorithm for all array sizes. This difference in efficiency can be linked to what each algorithm does to compare values to one another. Both algorithms gradually build up a sorted subarray on the left side of the array. The selection sort algorithm searches through all the elements in the remaining unsorted right side of the array to find the minimum value. The insertion sort algorithm, on the other hand, compares the current value to the value immediately to the left. This allows the insertion sort algorithm to make significantly less comparisons than its counterpart in most situations and thus sort an array much quicker. The fact that the selection sort algorithm needs to check every element in the unsorted portion of the array is supported by the graph of insertion vs selection sort with initially increasing data. Despite the data already being in increasing order, it can be seen that the number of lines needed for selection sort to execute is nearly identical to the number of lines needed for the initially random data and close to the number of lines needed for initially decreasing data. Regardless of the array's orientation, selection sort must check every value in the unsorted section of the array.