Sorting Algorithm	Runtime Results							
	n = 500		n = 2000		n = 7000		n = 13000	
	Descending	Random	Descending	Random	Descending	Random	Descending	Random
Bubble Sort	0.1275	0.1394	2.0727	1.7225	26.3391	24.7853	110.9059	93.9458
Selection Sort	0.0345	0.0772	0.4698	0.6611	5.7566	7.2843	24.0551	30.1394
Insertion Sort	0.0546	0.0727	0.8788	0.6156	11.3579	6.5301	48.2128	23.6655

BUBBLE SORT

Bubble sort is not a practical sorting algorithm when n is large. For example, with test data of 13000, Descending = 110.91 and Random = 93.95 which are the highest runtimes across all trials done. The best case performance for this is $O(n)$ wherein the data is nearly or perfectly sorted in ascending order however, worst case performance $O(n^2)$ which is typically data in descending order or data that requires a lot of swaps before properly sorted.

SELECTION SORT

Selection sort has $O(n^2)$ complexity which makes it inefficient on large lists. From the table above, as n increases, the runtime for it also increases. Moreover, this sort typically performs better than the insertion sort due to a smaller number of swaps in each pass but falls behind runtime performance when n reaches 13000. The runtime for each data size with two sort options are mostly equal since Best = Average = Worst case performance.

INSERTION SORT

Insertion sort is efficient for small data sets with complexity $O(n)$ but has a worst case performance $O(n^2)$ for data sets which needs the sort to go through all elements in each pass. For instance, the runtimes for descending or reversed options are slower than the random options. At n = 13000, Descending = 48.21 which is far slower than Random = 23.67 even with the same number of data to sort through.