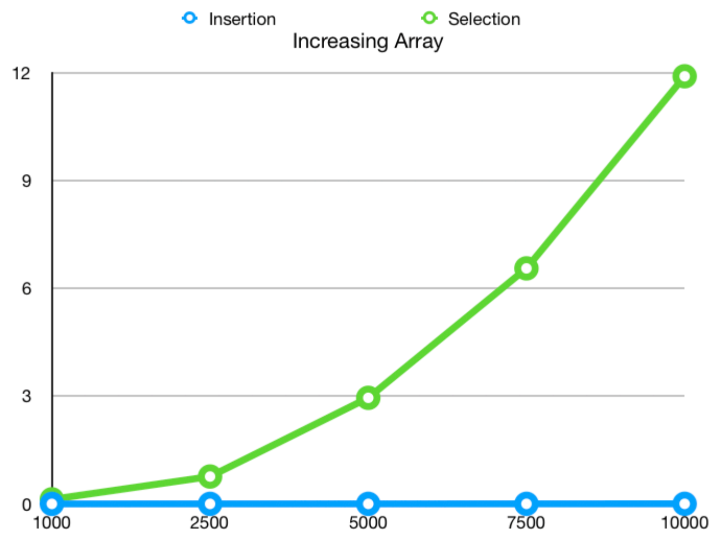
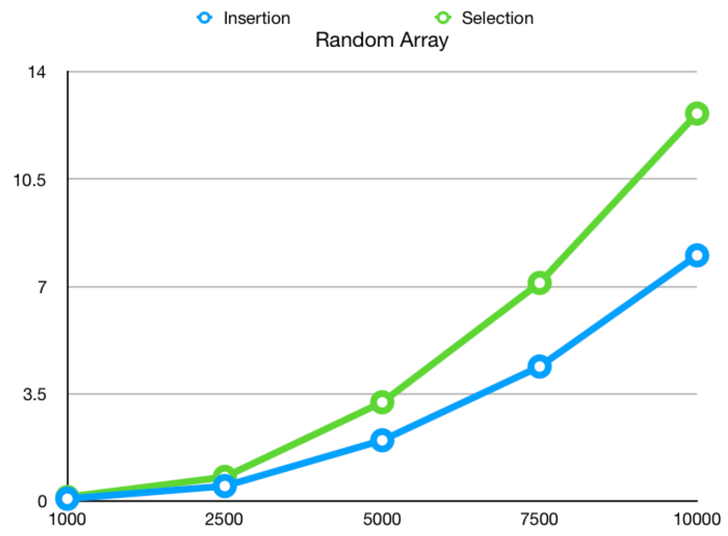
In this project I have analyzed the differences between insertion and selection sorting and how it impacts the processing time of arrays ranging from 1,000 to 10,000 cells. Each of the programs aims to sort the data within the arrays into increasing order. The insertion sort function works by checking if each number is in the correct position and swapping it when it is out of order. The selection sort function works by scanning the array for the smallest number and moving that number to the beginning.

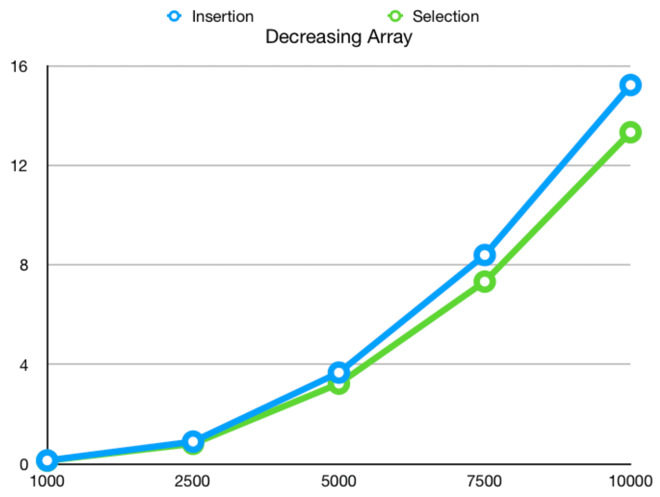
The differences in sorting times are the most obvious when looking at the arrays populated with already sorted numbers in increasing order. The insertion sort function sorted the data by looking at each number and moving it until it was in the correct position. This was very efficient in this case because every number was already in its correct position and the function only ran through one time for each number. The selection sort function was very inefficient because it had to look at every number in the array each time to find the next lowest value. This means that the processing time increased exponentially for the larger arrays. As seen by the graph, the insertion sort function processed all of the arrays in close to 0 seconds. The selection sort time increased up to 12 seconds for the largest array of 10,000.



Similar to the increasing array, the random array was processed faster by the insertion sort. However, the times were much closer with each being only a few seconds apart. The gap increased with the larger arrays showing that insertion sort is still more efficient in this case. Also, the selection sort times were very similar to the increasing array times. This is because the function is going through the same number of steps in both sets of arrays.



While the selection function was relatively inefficient for increasing and randomly ordered arrays. It was more efficient with the decreasing order array. The decreasing array represents the worst-case scenario, and this further demonstrates that the selection sorting time will be very similar no matter how the array is initially set up. Insertion sorting took much longer because none of the numbers were in the correct position and it had to individually move each cell to the correct position.



Overall, the insertion sort function was more efficient than the selection sort because it saves time when the numbers are already in their correct position. The selection sort ignores correctly positioned numbers and will scan through the entire array each time. Due to these differences the selection sort will usually take longer than insertion except for the worst-case scenario. In the worst case, selection swaps fewer numbers leading to the faster processing times. This means selection sort is more reliable and will be more consistent, but the insertion sort is almost always faster except for the worst case.

Increasing

	Insertion	Insertion	Insertion	Insertion	Insertion	Selection	Selection	Selection	Selection	Selection
1000	0.000296	0.000262	0.000270	0.000264	0.000284	0.136128	0.123870	0.125002	0.124136	0.125906
2500	0.000670	0.000624	0.000604	0.000606	0.000632	0.823432	0.747008	0.742926	0.749416	0.742914
5000	0.001274	0.001186	0.001268	0.001216	0.001184	3.144676	2.889026	2.919900	2.920820	2.879194
7500	0.001704	0.001776	0.001820	0.001784	0.001778	7.005298	6.428768	6.401988	6.474828	6.446072
10000	0.002398	0.002450	0.002380	0.002576	0.002430	12.158032	11.495198	12.753452	11.500056	11.567772

Decreasing

	Insertion	Insertion	Insertion	Insertion	Insertion	Selection	Selection	Selection	Selection	Selection
1000	0.15789	0.137494	0.136444	0.140176	0.137400	0.149844	0.127812	0.133634	0.127700	0.127782
2500	0.905402	0.886622	0.914572	0.897220	0.897148	0.850668	0.785530	0.812140	0.799642	0.806874
5000	3.770902	3.627198	3.626630	3.655816	3.687912	3.271414	3.194620	3.269472	3.180562	3.231976
7500	8.391858	8.294050	8.397932	8.442114	8.479356	7.374764	7.170982	7.536214	7.274768	7.297280
10000	14.765006	15.201354	15.600120	15.370278	15.226596	13.629884	13.108460	13.698266	13.099464	13.136924

Random

	Insertion	Insertion	Insertion	Insertion	Insertion	Selection	Selection	Selection	Selection	Selection
1000	0.079584	0.070984	0.072212	0.072274	0.072182	0.133506	0.125398	0.135672	0.129566	0.124498
2500	0.494108	0.477272	0.482264	0.516248	0.471102	0.786286	0.778040	0.817260	0.798566	0.751584
5000	1.941464	2.044158	1.934272	2.084418	1.897042	3.167204	3.435932	3.226256	3.199946	3.072710
7500	4.276324	4.439420	4.518622	4.469892	4.229464	6.930300	7.403632	7.562240	6.849078	6.824568
10000	7.680748	8.207738	8.356066	8.174932	7.638630	12.328820	12.928668	13.339046	12.449042	12.138982