Insertion sort and selection sort provided two different approaches to reordering an array of numbers in increasing order. Insertion sort sorted an array by comparing the current element with its previous elements. If the current element was greater than its previous adjacent element than they swapped spots. This was done repeatedly until the smallest elements got placed at the beginning of the array at each iteration. On the other hand, selection sort sorted an array by finding the minimum value at each iteration and placing it at the beginning of the array. These methods can be better explained with a deck of cards. For example, let's say there was a hand of cards with the card values being 3, 2, 1, 5, 4 in that order. To sort the cards using insertion sort, the process would start with the second card which would be the 2. It would then be compared with all the previous cards in the hand which in this case is only the 3. Since 2 was less than 3, they would switch places making the new hands 2, 3, 1, 5, 4. Then the next card in the hand (the 1) would need to be compared with the previous cards (3 and 2). Since the 1 was less than 3, they would swap places. Then since the 1 was less than the 2, they would also swap places making the new hand 1, 2, 3, 5, 4. This process would be repeated for the remaining cards in the hand until all of the cards are in order. For selection sort, let's imagine the same hand as the example in insertion sort (3, 2, 1, 5, 4). The process would start at the first card which was 3. Then one would look at the remaining cards to the right of 3 (2, 1, 5, 4) to find the smallest value compared to 3 which was 1 in this case. The 3 and 1 would swap making the new hand 1, 2, 3, 5, 4. Then the next value that would be swapped would be the 2. The remaining values to the right of 2 (3, 5, 4) would be checked to find the smallest value. In this case, 2 is the smallest compared to the remaining cards so there would be no swapping making the hand (1, 2, 3, 5, 4). This process was repeated for the remainder of the cards until the cards have been completely sorted.

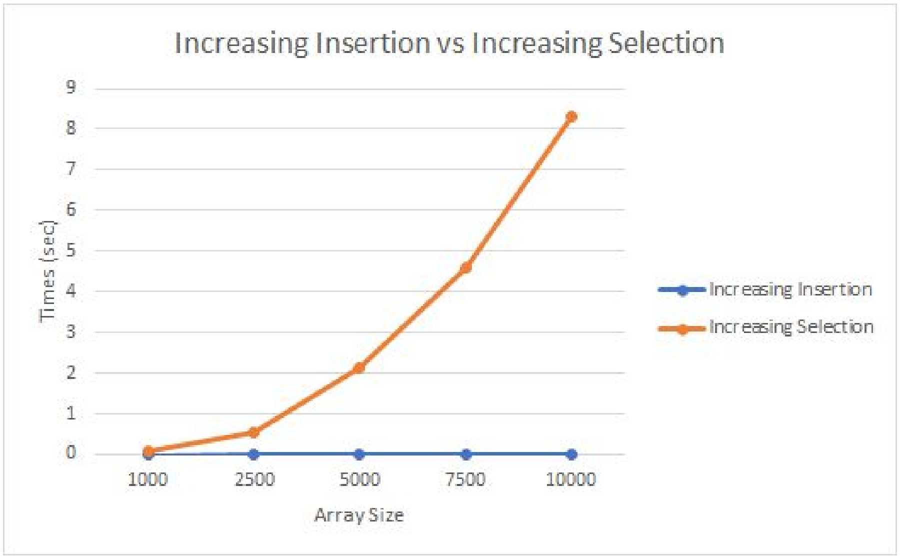
Each of these methods took varying times to sort arrays as shown with the graphs for increasing, decreasing, and random arrays. Moreover, comparing the times taken to sort increasing, decreasing, and random array, insertion sort proved to be better than selection sort overall.

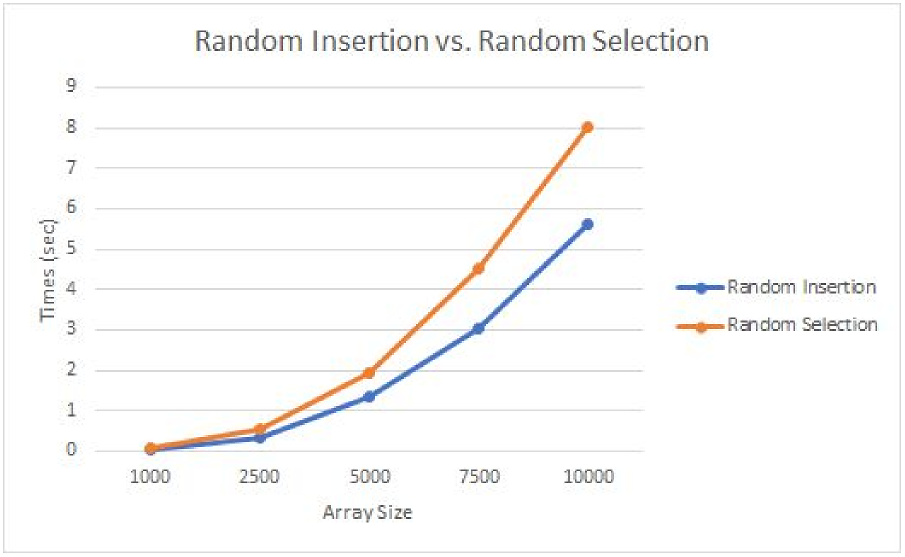
For sorting an increasing array (already sorted), there was a huge difference in time to sort between the two different sorting methods. The graph showed that the time for sorting an increasing array using selection sort was significantly higher than the time for sorting an increasing array using insertion sort at each of the varying lengths. Moreover, the rate of increase in time was also higher using selection sort; the selection sort plot increased parabolically while the insertion sort plot increased linearly. This occurred because the insertion sort can skip the steps it takes to compare and swap elements since the array has already been sorted. This made it so that the time taken to sort the array only came from the time to iterate through the whole array. On the other hand, in addition to iterating through the whole array, selection sort had to find the minimum value every iteration which contributed to the addition of numerous unnecessary steps. This combination of having the find the minimum value and iterating through the array was what made it take more time than insertion sort.

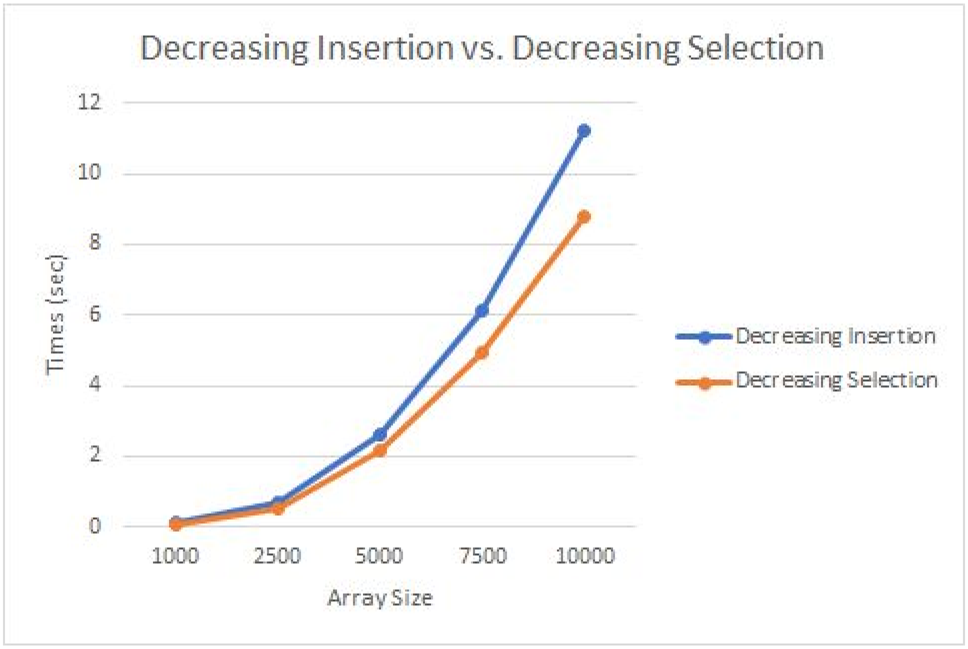
For sorting a decreasing array, the time difference between selection sort and insertion sort was relatively minute with selection sort taking slightly less time than insertion sort. Both plots increased parabolically, but insertion sort increased at a slightly faster rate than selection sort. This happened because a decreasing time array was the worst-case scenario (takes the most time) for insertion sort. At each iteration, insertion sort will always have to swap all the elements previous to it due to the decreasing order of the array. On the other hand, selection sort only has to check to find the minimum value every iteration but never has to swap any values. This is because the minimum value is already found without even having to check the remaining elements in the array. This greatly reduces the number of steps required to complete the method and is what makes selection sort faster than insertion sort for sorting decreasing arrays.

For sorting a random array, selection sort took more time than insertion sort although both increased parabolically according to the graph. Moreover, selection sort increased at a slightly higher rate than insertion sort. This occurred because in some cases insertion sort only required a single comparison and no swapping if the previous element was already less than the current element. However, the selection sort always has to iterate through all the remaining elements and find the minimum value. Insertion sort's ability to take fewer steps when necessary in certain cases compared to selection sort having to take a set number of steps each iteration is what made insertion sort faster than selection sort for sorting a random array.

In conclusion, insertion sort overall seemed to be the better sorting method compared to selection sort. This was shown by insertion sort drastically outperforming selection sort in when sorting an increasing array. Moreover, insertion sort was also better than selection sort when sorting a random array. Although selection sort slightly outperformed insertion sort when sorting a decreasing array, this only occurred because a decreasing array was a worst-case scenario for insertion sort.


Increasing Insertion vs Increasing Selection
9
8
7
6
5
4
3
2
1
0
Times (sec)
1000
2500
5000
7500
10000
Array Size
Increasing Insertion
Increasing Selection


Random Insertion vs. Random Selection
9
8
7
6
5
4
3
2
1
0
Times (sec)
1000
2500
5000
7500
10000
Array Size
Random Insertion
Random Selection



	IncIns #1	IncIns #2	IncIns #3	IncIns #4	IncIns #5	Avg
1000	0.000147	0.000152	0.000147	0.000152	0.000147	0.000149
2500	0.000367	0.000367	0.000368	0.000367	0.000366	0.000367
5000	0.000741	0.000734	0.001829	0.000754	0.000741	0.00096
7500	0.001107	0.001111	0.0011	0.001105	0.001105	0.001106
10000	0.001474	0.001476	0.001484	0.001472	0.001475	0.001476

	DecIns #1	DecIns #2	DecIns #3	DecIns #4	DecIns #5	Avg
1000	0.086395	0.102586	0.107464	0.102093	0.115516	0.102811
2500	0.588537	0.701291	0.769478	0.723886	0.657092	0.688057
5000	2.73545	2.599392	2.619361	2.454061	2.664461	2.614545
7500	6.138035	6.304264	6.590779	5.877043	5.781249	6.138274
10000	10.85962	11.48201	11.31326	11.04836	11.37852	11.21635

	RanIns #1	RanIns #2	RanIns #3	RanIns #4	RanIns #5	Avg
1000	0.044452	0.044689	0.051771	0.043436	0.04348	0.045566
2500	0.290474	0.360573	0.370167	0.389349	0.291195	0.340352
5000	1.192992	1.515004	1.228517	1.20778	1.501439	1.329146
7500	3.59084	3.01454	2.724219	2.790674	3.076527	3.03936
10000	6.488445	5.267067	5.650495	5.343455	5.233676	5.596628

	IncSel #1	IncSel #2	IncSel #3	IncSel #4	IncSel #5	Avg
1000	0.071033	0.108503	0.109249	0.071132	0.069512	0.085886
2500	0.441503	0.635989	0.550602	0.548424	0.438504	0.523004
5000	1.833987	1.953516	1.93123	1.966076	3.030445	2.143051
7500	4.690254	4.10622	4.97617	4.528232	4.733117	4.606799
10000	8.710955	8.910027	7.680119	8.061683	8.187305	8.310018

	DecSel #1	DecSel #2	DecSel #3	DecSel #4	DecSel #5	Avg
1000	0.082297	0.087375	0.073803	0.083025	0.099429	0.085186
2500	0.448964	0.550766	0.63334	0.44668	0.596489	0.535248
5000	2.277547	2.327874	2.444878	1.826341	2.032465	2.181821
7500	4.829837	4.569791	5.150984	4.629283	5.495112	4.935001
10000	8.56036	7.918562	8.991836	8.779669	9.607059	8.771497

	RanSel #1	RanSel #2	RanSel #3	RanSel #4	RanSel #5	Avg
1000	0.07143	0.116728	0.070458	0.105726	0.106091	0.094087
2500	0.441745	0.556385	0.620873	0.483448	0.622692	0.545029
5000	2.048795	2.158816	1.764513	2.019436	1.775725	1.953457
7500	4.400012	4.493824	4.291193	4.362436	5.012538	4.512001
10000	8.04776	8.033959	7.56455	8.249232	8.218074	8.022715