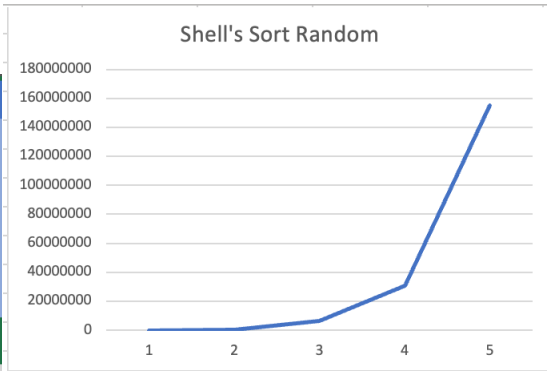
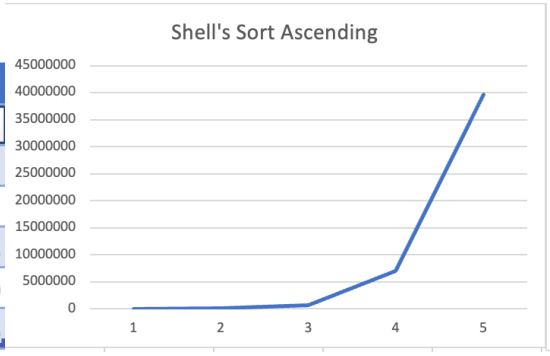
Program 2 Analysis

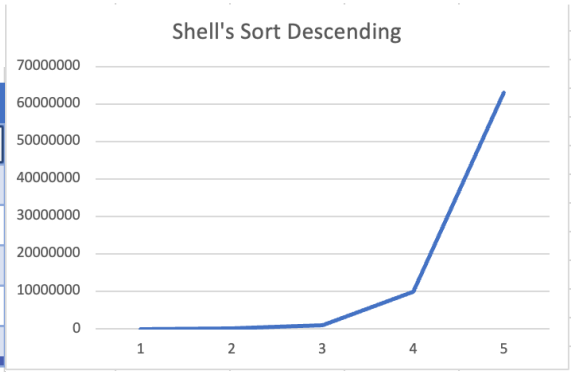
Shell's Sort Random		
Size of n	Time to Complete	Factor of Change
100	29417	
1000	432792	14.7
10000	6196875	14.3
100000	30883250	4.9
1000000	155377583	5



Shell's Sort Ascending		
Size of n	Time to Complete	Factor of Change
100	10084	
1000	146583	14.5
10000	704417	4.8
100000	7065833	10
1000000	39736125	5.6

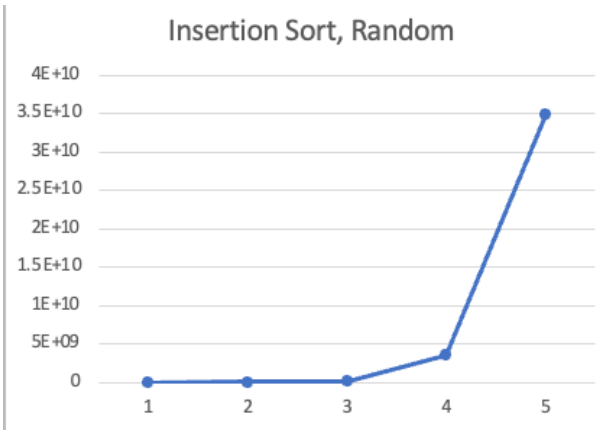


Shell's Sort Decending		
Size of n	Time to Complete	Factor of Change
100	17333	
1000	245083	14.1
10000	1106708	4.5
100000	9891875	8.9
1000000	62933458	6.3

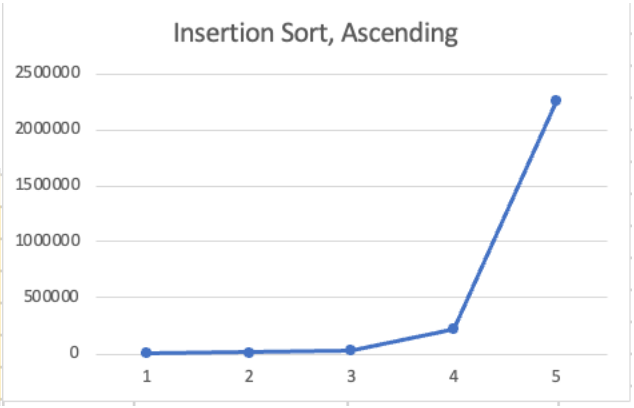


When running Shell's sort, the first thing I noticed was that it ran faster than the other functions. The function's order of growth is O(n^2). When looking at the table you can see how the factor of growth starts off huge as it the graph takes off exponentially and decreases and levels out as the climb of the graph becomes a little more regular. If I had the time and space on my machine I would like to run the same algorithm again but rather than stopping k at 6 I would like to take k to 10 or 15 to see what the factor increase is then. Overall learning to work Shell's sort and piecing the algorithm together with information from the text along with research from other sources, I would say it was the most fun to work through and see the empirical results from as I have never worked with this algorithm before.

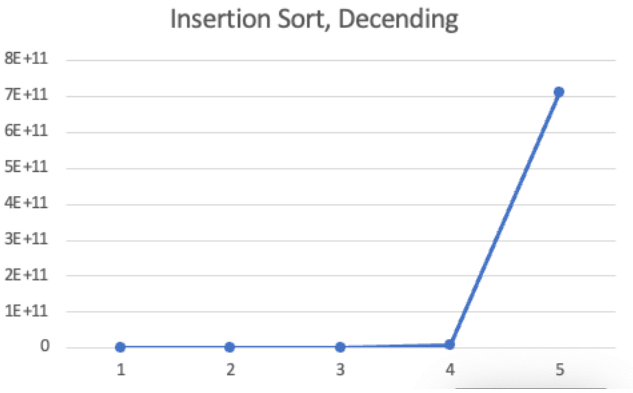
Insertion Sort, Random		
Size of n	Time to Complete	Factor of Change
100	32250	
1000	2119125	65.7
10000	57433667	27.1
100000	3438898750	5.99
1000000	34859391167	1.01



Insertion Sort, Ascending		
Size of n	Time to Complete	Factor of Change
100	1833	
1000	11875	6.5
10000	27125	2.3
100000	221792	8.1
1000000	2258792	10.2

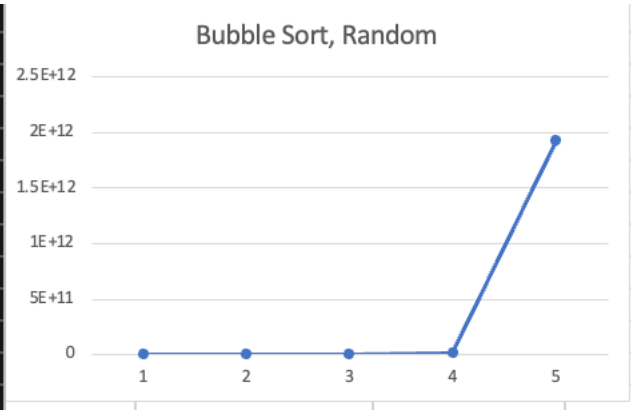


Insertion Sort, Decending		
Size of n	Time to Complete	Factor of Chang
100	57833	
1000	3718916	64.3
10000	71341417	19.2
100000	6920143042	97
1000000	7.11528E+11	102.8

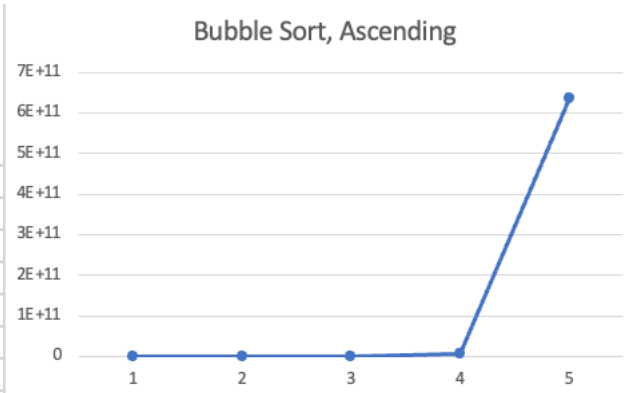


Insertion sort was the next algorithm that I ran, and it had an O(n^2) order of growth as well. This is demonstrated neatly in the graph above. The graph shows exponential growth as usually expressed by a quadratic function. As far as empirical complexity, this graph also ran quite fast compared to the later algorithms. Whether that is because I ran it second, or due to the complexity of the algorithm itself, I guess that would depend on further research and my machine. I think once again, that if I had more time and resources I would run this test again, but on a different machine and organize the algorithms all in a different order to see if it would change the run times. I run on a Mac M1 chip. I think if I were to do the experiment again I would do it on a machine with a slower CPU and see how that changes the data.

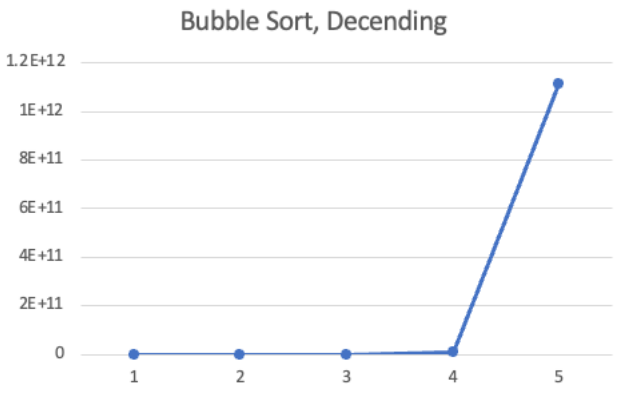
Bubble Sort, Random		
Size of n	Time to Completion	Factor Increase
100	92666	
1000	3590917	38.8
10000	154550500	43
100000	18792302750	121.6
1000000	1.92225E+12	102.3



Bubble Sort, Ascending		
Size of n	Time to Completion	Factor Increase
100	41333	
1000	1915583	46.3
10000	62629500	32.7
100000	6260450667	100
1000000	6.36991E+11	101.7

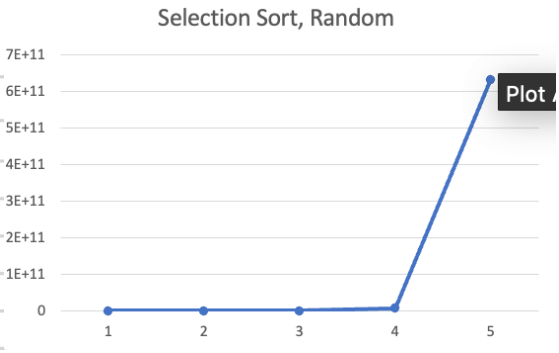


Bubble Sort, Decending		
Size of n	Time to Completion	Factor Increase
100	80958	
1000	3293959	40.7
10000	107485750	32.7
100000	10885836417	101.3
1000000	1.11155E+12	102.1

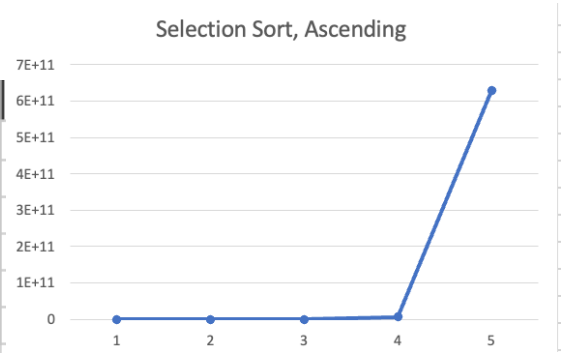


The next algorithm ran was bubble sort. When ran, bubble sort ran slower than Shell's sort and insertion sort, but it ran in a similar time to selection sort. It has an order of growth of O(n^2). It had a similar speed to the first two during the first three runs, but when k = 5 I noticed the change in speed. Of course, when k = 6 the time running it was also slow, but those changes were displayed by every function ran, naturally as n increased so would the time. Something that I noticed with bubble sort was that it took the longest to complete if you compare all the numbers of the tables.

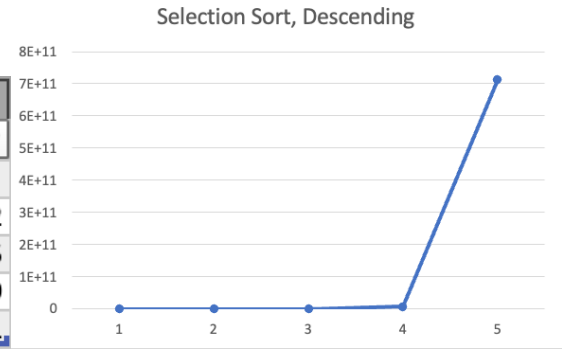
Selection Sort, Random		
Size of n	Time of Completion	Factor Increase
100	50667	
1000	2010166	39.7
10000	63087750	31.4
100000	6255976000	99.2
1000000	6.32407E+11	101.1



Selection Sort, Ascending		
Size of n	Time of Completion	Factor Increase
100	41958	
1000	1609000	38.3
10000	62674667	39
100000	6260125708	99.9
1000000	6.28157E+11	100.3

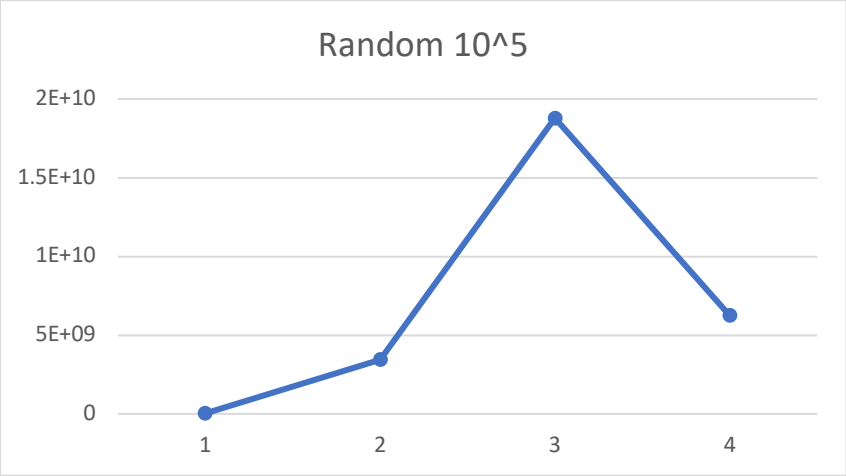


Selection Sort, Descending		
Size of n	Time of Completion	Factor Increase
100	49333	
1000	1785417	36.2
10000	70582334	39.5
100000	7049882875	99.9
1000000	7.12834E+11	101.1



Selection sort being the last algorithm took my machine the longest to get through, but its times were not larger than the algorithms that came before it. My thought process is that it must have been something to do with my machine, but I am not sure. The order of growth for this function is O(n^2), the same as the other functions. Something that draws the eye with this graph is how exponential this function is. The graph itself nearly looks like an obtuse angle with the first four runs not taking nearly the amount of time as k = 6. As I said previously, if I had the opportunity and more resources I would like to do this experiment all over again on another machine with larger values of k to see what I would get.

Special Case Discussion:



1 = Shell, 2 = Insertion, 3 = Bubble, 4 = Selection

When looking at the chart presented it becomes clear that one of the points took quite a bit longer to run than the rest. I thought this was strange considering all the functions had the same time complexity. Whether this was my machine or not is still up to debate. I was running this program late at night with no applications open at the time. I have no idea why it took so much longer to run bubble sort than the rest of the algorithms. Maybe it had to do with the way the algorithm was coded but looking back through lecture notes from not only 2400, but also 1310 I do not think I made a mistake on the code itself. I would say Bubble sort acts as a special case considering how much of an outlier it presents as on the graph above.

Summary

One thing I noticed over all that I expected to be different was the factor of change or the factor increase. I expected it to be linear for every function and it was not. Sometimes there would be a very large spike like in the random and descending tables for the insertion sort. These large spikes I feared might have skewed my data, but it was the average of my runs, so now I am left to wonder if it was my machine. It was interesting to see how fast all the sorting algorithms ran. In class it has always been discussed but getting to run the algorithms and code oneself it is easier to see the empirical data. It is also interesting to see when discussing with others how the time differs between machines. While one machine might take 10 hours or more to run an algorithm another can take only 3. This was a cool lab to experiment with and, if given the time and resources, would be fun to continue playing with all the different algorithms to see more possible outcomes.

NOTE: In all tables the time measurement used is nanoseconds, due to the calculation of the graph, the abbreviation had to be taken out.