Ephraim Sun

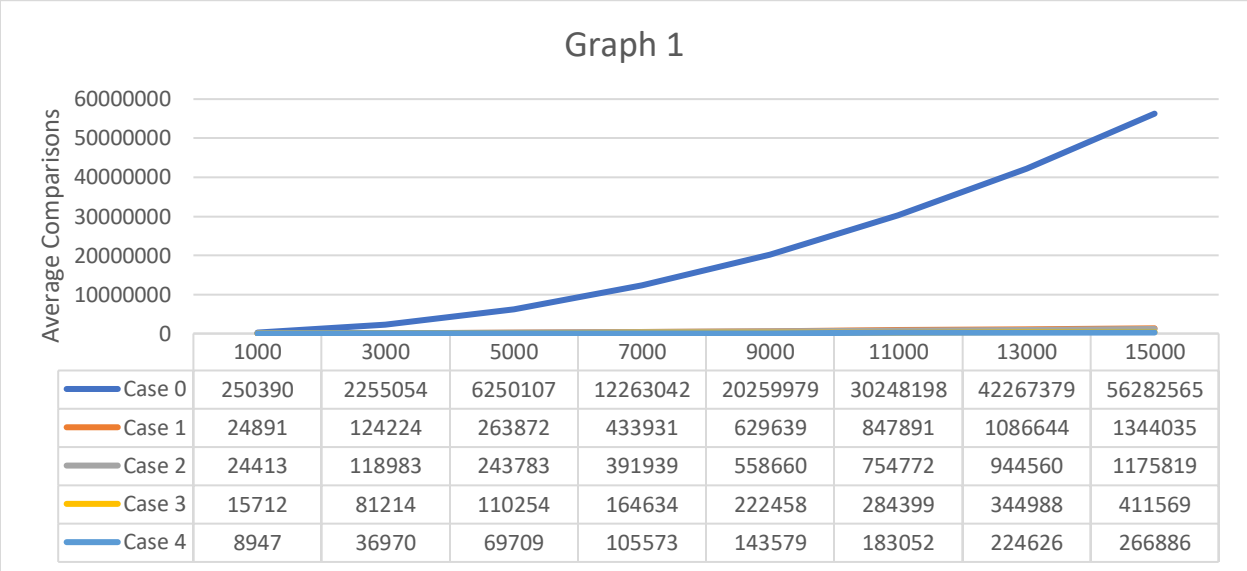
Project 1 Analysis

After careful analysis of Shell Sort by generating 100 different sets and taking the average as well as the standard deviation where n = 1000, 3000, 5000, 7000, 9000, 11000, 13000, and 15000, we observe the following graphs and data points.

N	1000		3000		5000		7000		9000		11000		13000		15000	
Case	Average	SD	Average	SD	Average	SD	Average	SD	Average	SD	Average	SD	Average	SD	Average	SD
0	250389.6	5645.876361	2255054.46	25726.00816	6250107.4	63538.05906	12263042.24	87664.25024	20259978.9	148007.303	30248197.5	179820.41	42267378.8	258752.053	56282565	296574.959
1	24890.96	99.69352236	124224.4	195.164341	263872.17	318.5176308	433931.06	350.9252861	629638.88	444.597555	847891.36	520.207988	1086643.67	639.112401	1344034.68	664.197198
2	24413	2687.865119	118982.63	12685.19911	243782.51	27746.05584	391939.35	44389.52671	558659.84	59152.008	754772.05	98714.118	944560.04	105532	1175818.77	140755.706
3	15712.22	346.632733	81213.88	216726.5362	110254.44	2666.855535	164634.21	4447.706641	222457.9	5644.0331	284399.16	7946.41385	344987.66	7510.90637	411568.57	10549.3175
4	8946.83	251.0907826	36969.96	810.7617643	69709.01	1227.55529	105572.82	1749.278219	143579.21	2390.84254	183052.38	2847.17468	224626.06	3514.35638	266885.79	3674.9367

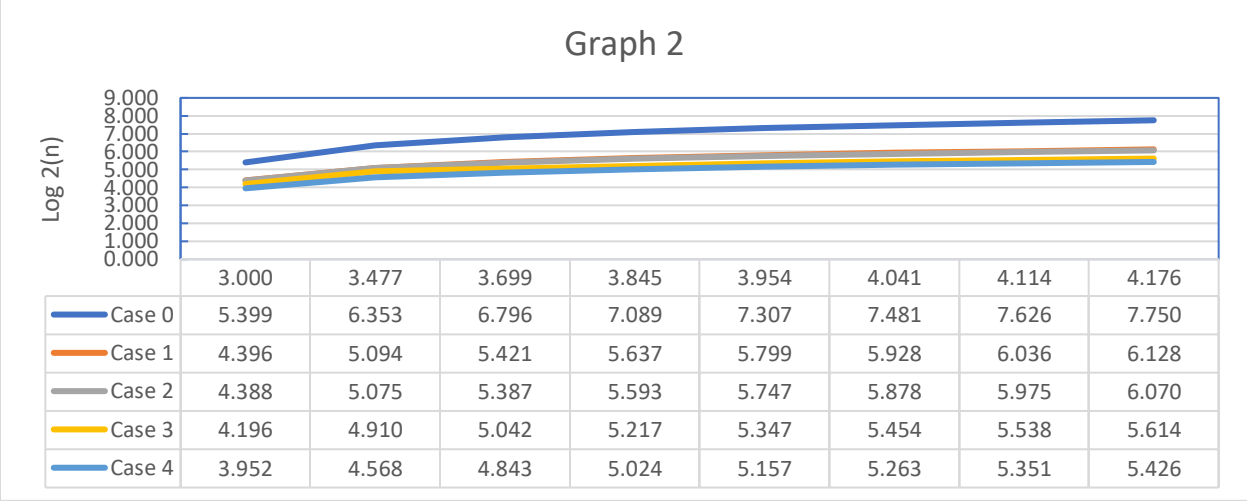
	1000	3000	5000	7000	9000	11000	13000	15000
Case 0	250390	2255054	6250107	12263042	20259979	30248198	42267379	56282565
Case 1	24891	124224	263872	433931	629639	847891	1086644	1344035
Case 2	24413	118983	243783	391939	558660	754772	944560	1175819
Case 3	15712	81214	110254	164634	222458	284399	344988	411569
Case 4	8947	36970	69709	105573	143579	183052	224626	266886

Based on the following graph, we can see Case 0 takes the most number of comparisons, making it the least efficient, ~ O(n^2). Moreover, we can see that Case 1, Case 2, Case 3, and Case 4 are much closer in comparisons on Graph 1, since they are Shell Sort Algorithm and roughly O(nlogn). Thus, we need to take a closer look.



	3.000	**3.477**	**3.699**	**3.845**	**3.954**	**4.041**	**4.114**	**4.176**
Case 0	**5.399**	**6.353**	**6.796**	**7.089**	**7.307**	**7.481**	**7.626**	**7.750**
Case 1	**4.396**	**5.094**	**5.421**	**5.637**	**5.799**	**5.928**	**6.036**	**6.128**
Case 2	**4.388**	**5.075**	**5.387**	**5.593**	**5.747**	**5.878**	**5.975**	**6.070**
Case 3	**4.196**	**4.910**	**5.042**	**5.217**	**5.347**	**5.454**	**5.538**	**5.614**
Case 4	**3.952**	**4.568**	**4.843**	**5.024**	**5.157**	**5.263**	**5.351**	**5.426**

In Graph 2, we took the log base 10 of both N and the average number of comparisons, showing the performance of all the graph. As shown below, see that Case 4 seems to consistently perform better than the other cases.



In conclusion, through our program and subsequent analysis, we put into action both Insertion Sort and Shell Sort and conducted an examination of their respective time complexities. Initially, we established that, in most instances, Shell Sort outperforms Insertion Sort in terms of efficiency, especially when dealing with a completely random set of numbers. We experimented with four distinct gap sizes in Shell Sort to illustrate the influence of gap size on the sorting of an unsorted array. The efficiency can vary with different gaps, contingent on the data size. Nevertheless, in this instance, we discovered that Case 4 yielded the highest efficiency, as determined by the average number of comparisons.