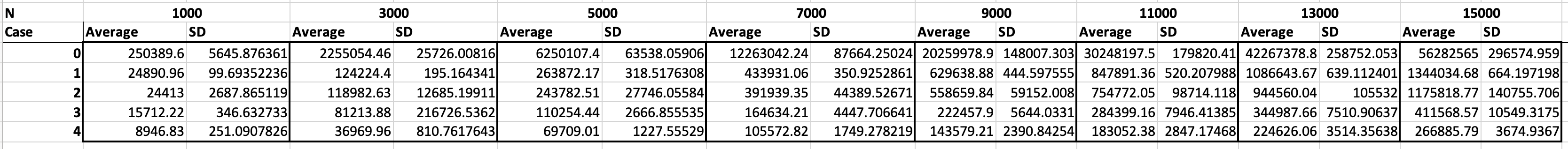
**Project 1 Analysis**

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.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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.