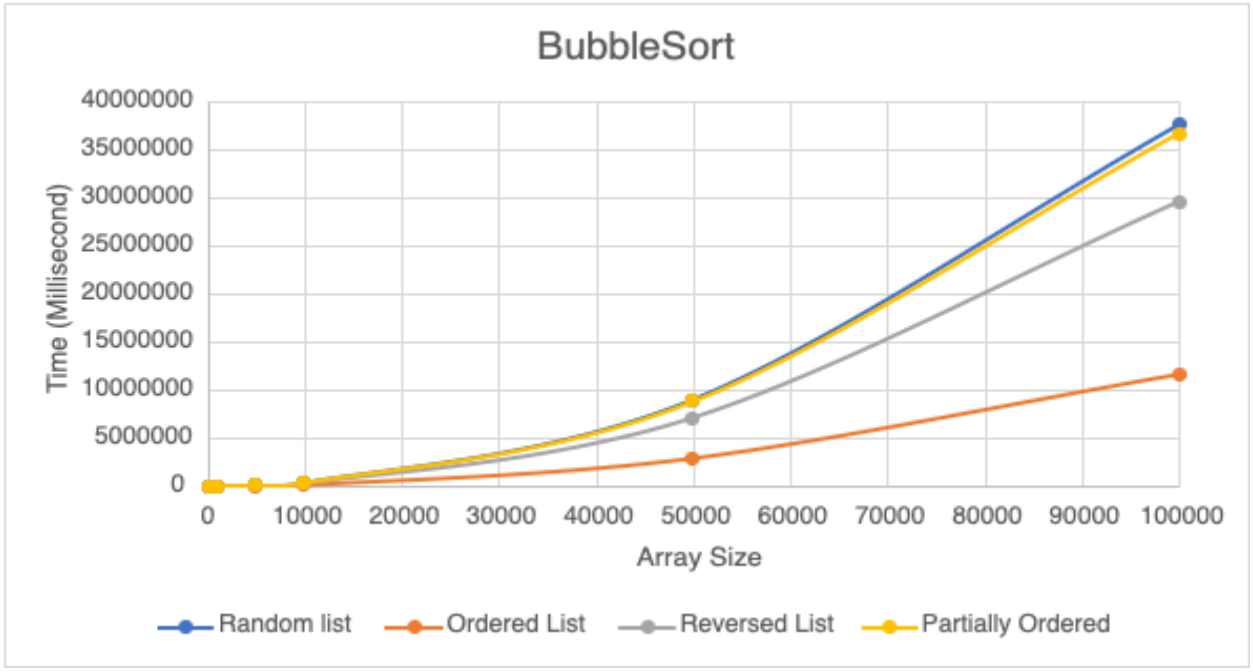
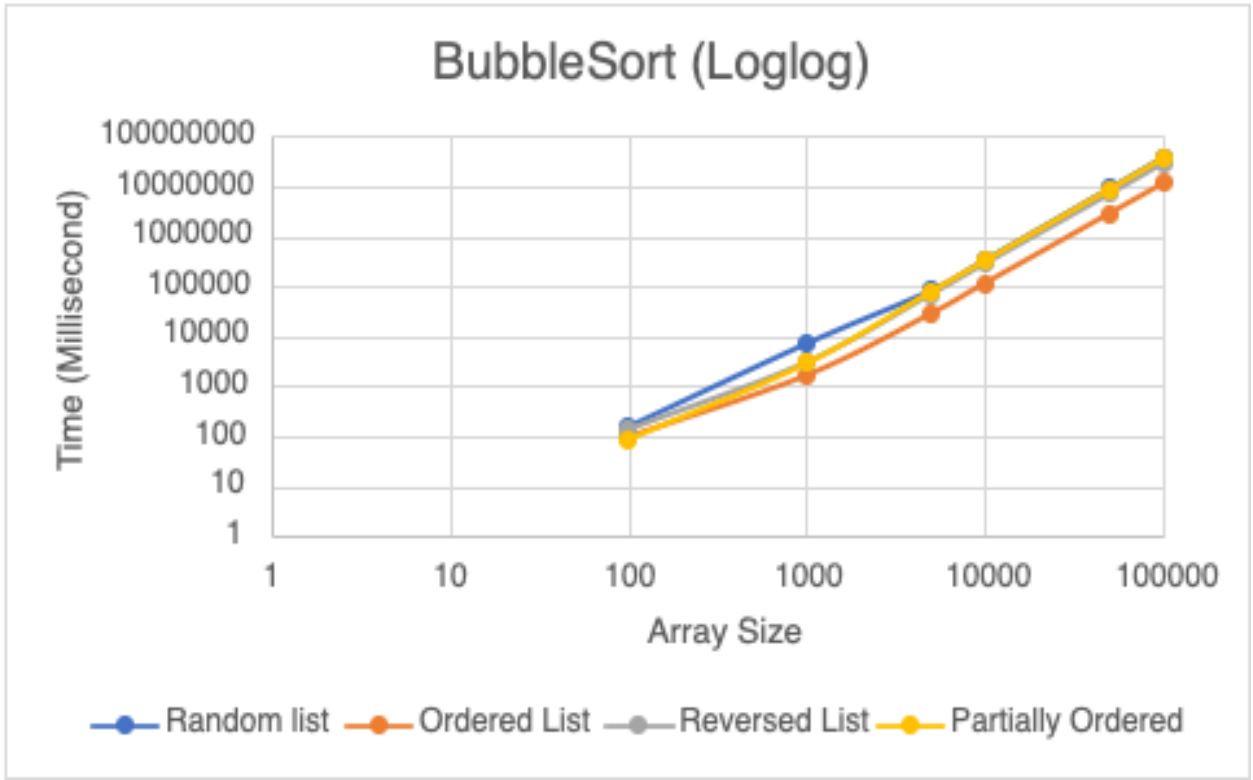
Assignment #3 Report

Anushka Chougule

NOTE: In the following assignment I have increased the data up till 100K Array Size due to computer runtime/ crashing issue. However this did not impact my data collecting methods nor the pattern of the data.

Bubble SORT:





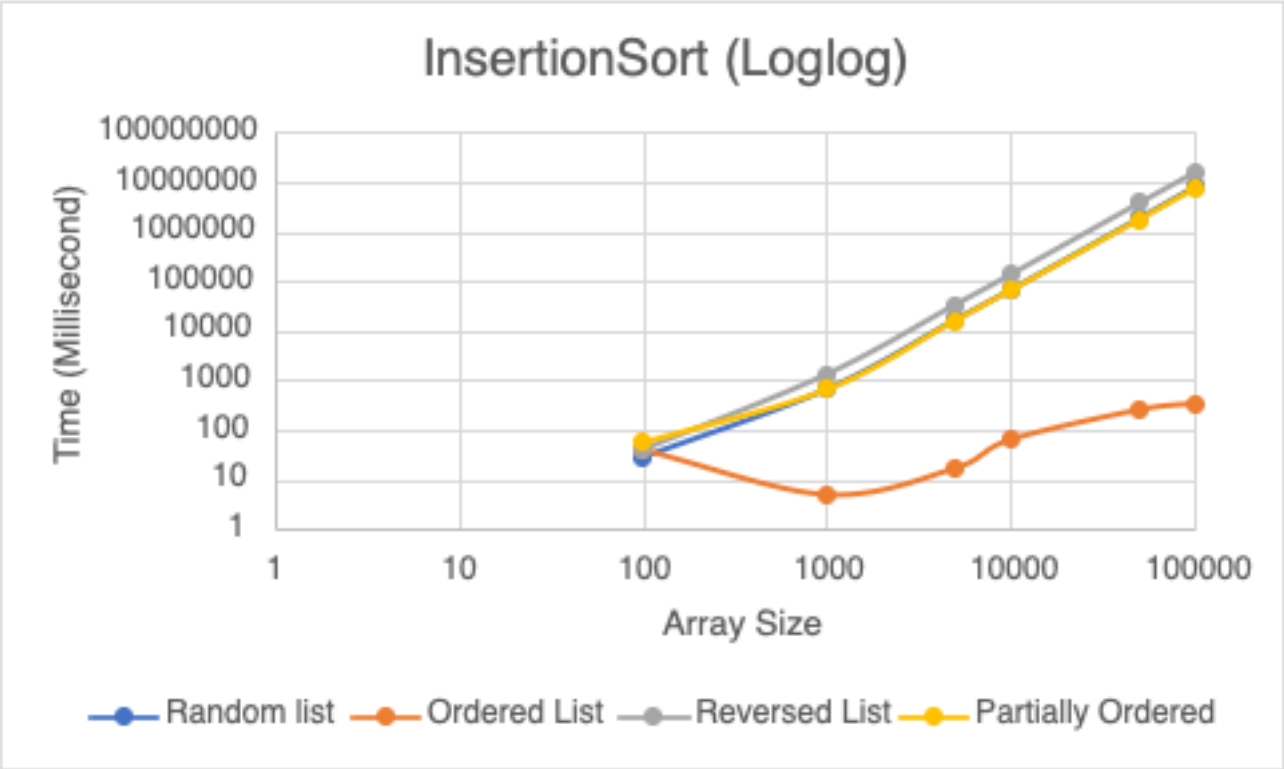
BubbleSort:

- **BEST Case: O(n)** - when the list is already sorted in this case Ordered List
- **AVERAGE Case: O(n²)** - when the list needs to be sorted (Reversed List)
- **WORST Case: O(n²)** - when the list needs to be sorted (Random List/Partially Ordered)

  Space complexity : O(1)

The tables above are an example of BubbleSort being tested using 4 different tests (random, ordered, reversed and particularly order lists). Each Array falls within either best, average or worst case and understanding which array falls into which case is necessary when building a program as it can lower the program execution time. The BubbleSort table shows the lowest program execution time was Ordered List and the highest was Random/partially ordered lists.

INSERTION SORT:



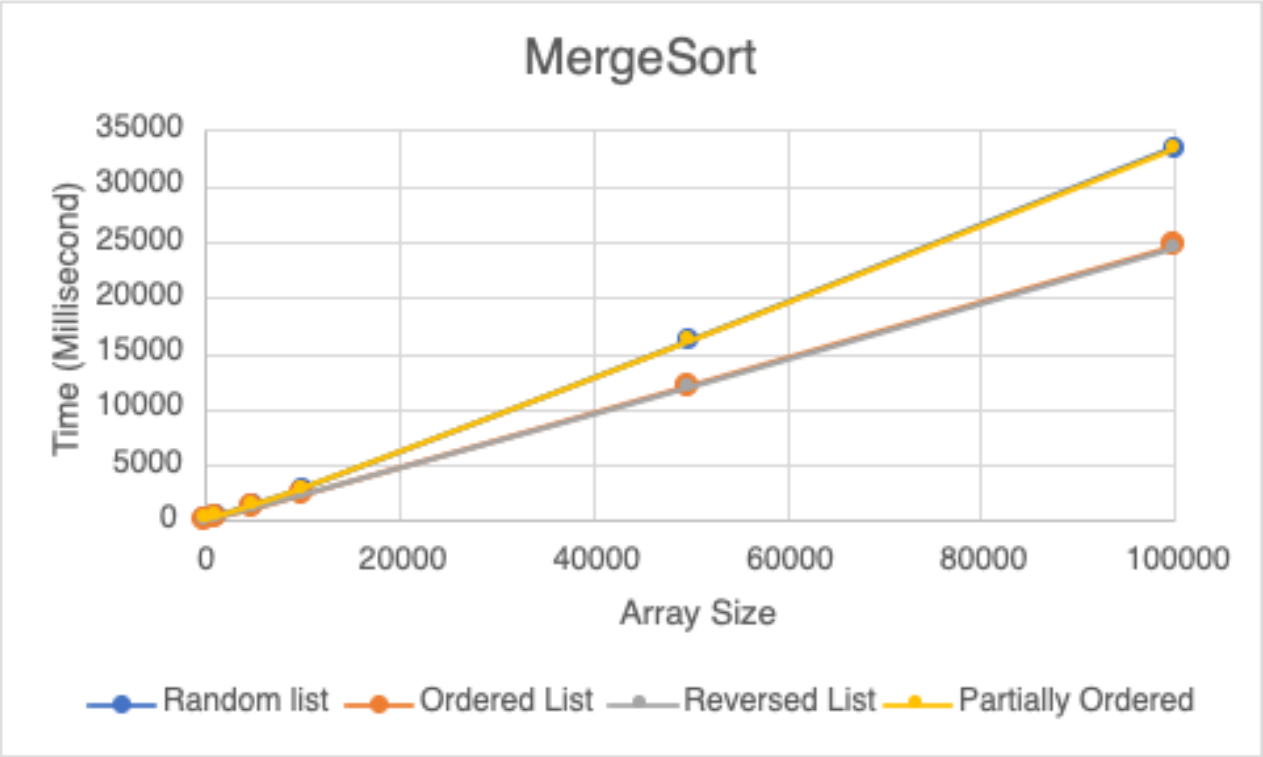
InsertionSort:

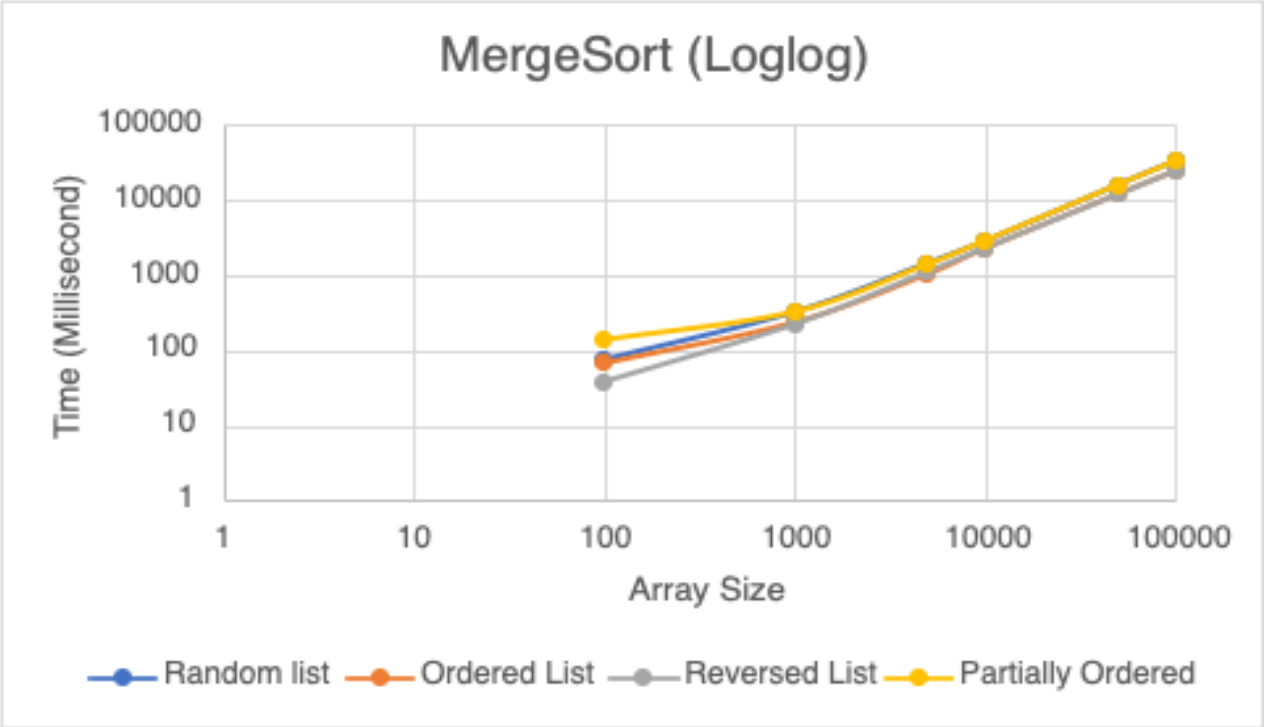
- **BEST Case: O(n)** - when the list is already sorted in this case Ordered List
- **AVERAGE Case: $O(n^2)$** - when the list needs to be sorted (Random List/Partially Ordered)
- **WORST Case: $O(n^2)$** - when the list needs to be sorted (Reversed List)

  Space complexity: O(1)

The tables above are the time it took to execute the InsertionSort using the Ordered, random, reversed and partially ordered lists. These tests were necessary to run in order to better understand which would be the best case and what case to avoid/be careful of when programming. From the IntertionSort and IntertionSor t(Loglog) graph we can understand that the Ordered list took the least amount of time to run while on the other hand the order that took the most longest to run was the Revered list.

MERGE SORT:





MergeSort:

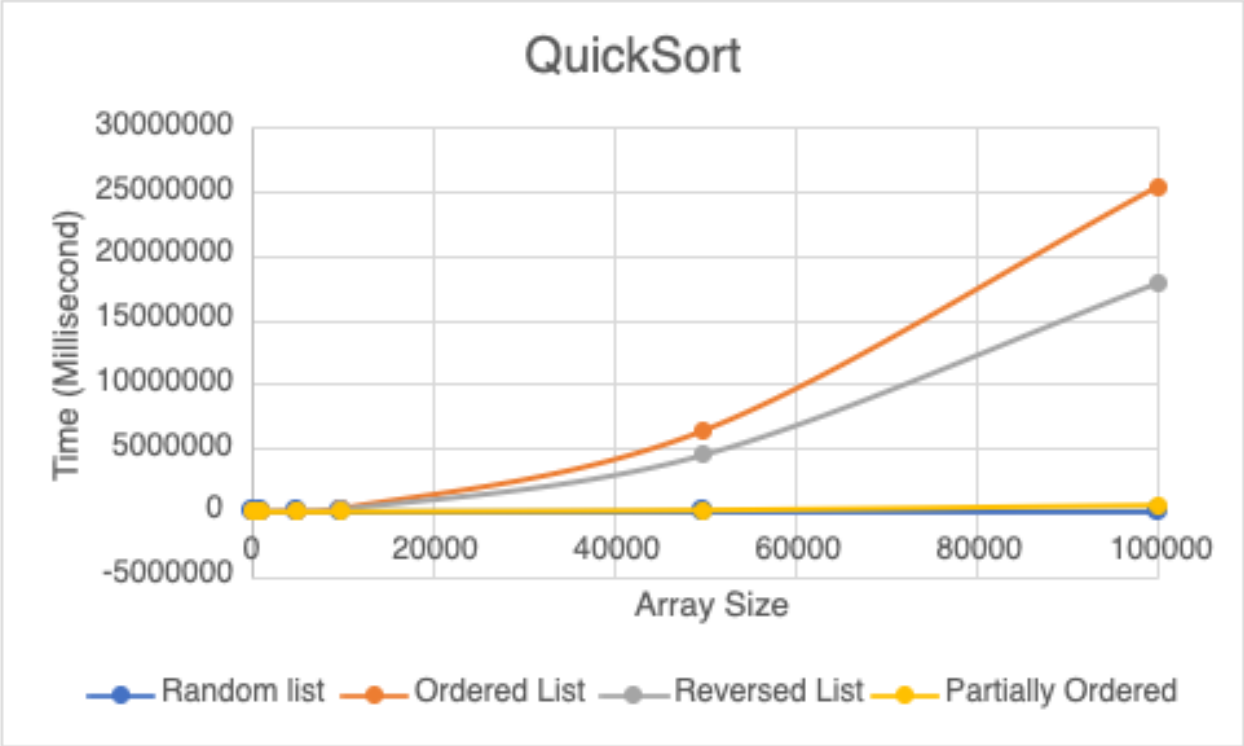
- **BEST Case: O(n*log n)**
- **AVERAGE Case: O(n*log n)**
- **WORST Case: O(n*log n)**

Space Complexity: O(n)

*Note: As shown in the graph above in the Merge Sort graph the order that took the least amount of time was ordered and reversed list while random and practically random took the longest amount of time.

The MergeSort table indicates the sorting list that took the least amount of time to compile was Reversed and Ordered lists. While the lists that took the longest amount of time to compile were Partially ordered and random lists. This data is incredibly important to comprehend as learning about these time complexities can help us better execute the array size in the fastest way possible.

QUICK SORT:



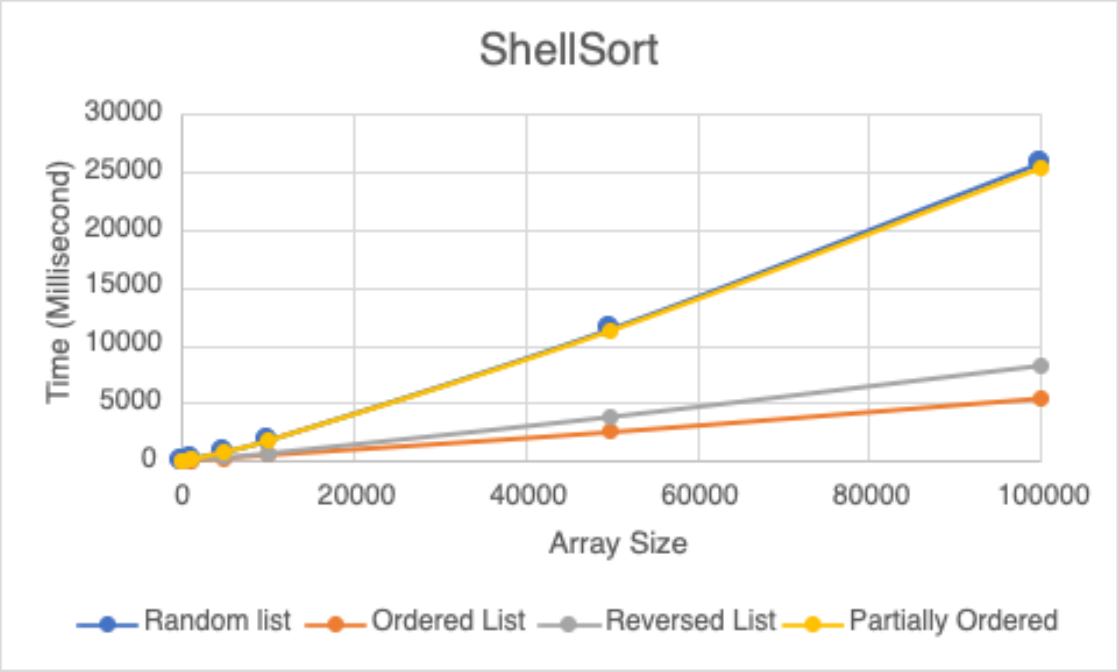
QuickSort:

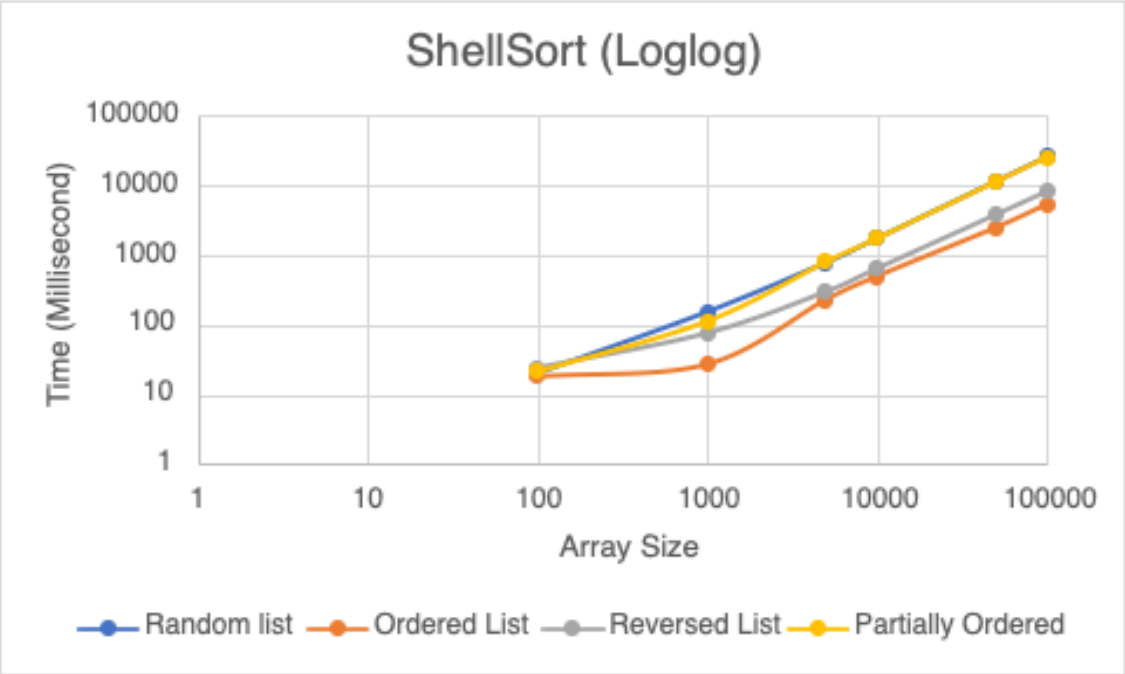
- **BEST Case:** **O(n*logn)**
- **AVERAGE Case:** **O(n*logn)**
- **WORST Case:** **O(n^2)**

  Space complexity: O(log n)

The graph above is an example of QuickSort execution runtime using various lists including Random list, Ordered list, reversed list and partially ordered lists. According to the constructed data the list that took the least amount of time to run was partially random lists while on the other hand the lists that took the longest to run were ordered and reversed lists. This data helps us understand the time complexity of the QuickSort function better, as well as the various cases (best, average and worst).

SHELL SORT:





ShellSort:

- **BEST Case: O(n log n)**
- **AVERAGE Case: O(n log n)**
- **WORST Case: O(n2)**

  Space Complexity: O(1)

The data collected for ShellSorts runtime for various array sizes shows the fastest list to run was ordered and reversed lists while the lists that took the longest to run were random and partially ordered lists. This graph helps us better understand which lists are going to take the least amount of time to execute while which one might take longer to run as the array size starts to increase.

Discuss the obtained convergence rates. For example, if two methods are O(n logn) but one is slower than the other, provide an explanation:

In this sorting algorithm the Merge Sort, Shell Sort and Quicksort all are O(n log n), however if you notice the time for each of this algorithm as the array size increases or even so the best case (lowest time complexity) you can notice there is a significant difference. This is because the way these algorithms order/arrange each of the listed lists (random, ordered, reversed and particularly order lists) varies immensely. For example MergeSort arranges the list by splitting it into smaller subarrays and then ordering them while on the other hand QuickSort arranges the list by using a pivot which recursively keeps arranging the code until fully sorted.This shows that as the size of the array increases for an O(n log n) algorithm the higher the time complexity.

This also applies to other sorting algorithms such as Bubble Sort and Insertion Sort which have the time complexity of O(n^2). Additionally both of these algorithms have nested loops as well as swap values until the array list is sorted.

Conclusion:

To sum up everything that has been stated, all five of the Sorting algorithms (BubbleSort, InsertionSort, MergeSort, ShellSort, QuickSort), have the same cases they differ within time complexity due to their specific methods they use to order the lists into an organize format.

Sources:

1. https://builtin.com/data-science/bubble-sort-time-complexity
2. https://developer.nvidia.com/blog/insertion-sort-explained-a-data-scientists-algorithm-guide/#:~:text=The%20worst%2Dcase%20(and%20average,O(n)%20time%20complexity.
3. https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-merge-sort/
4. https://www.javatpoint.com/quick-sort
5. https://www.geeksforgeeks.org/shellsort/