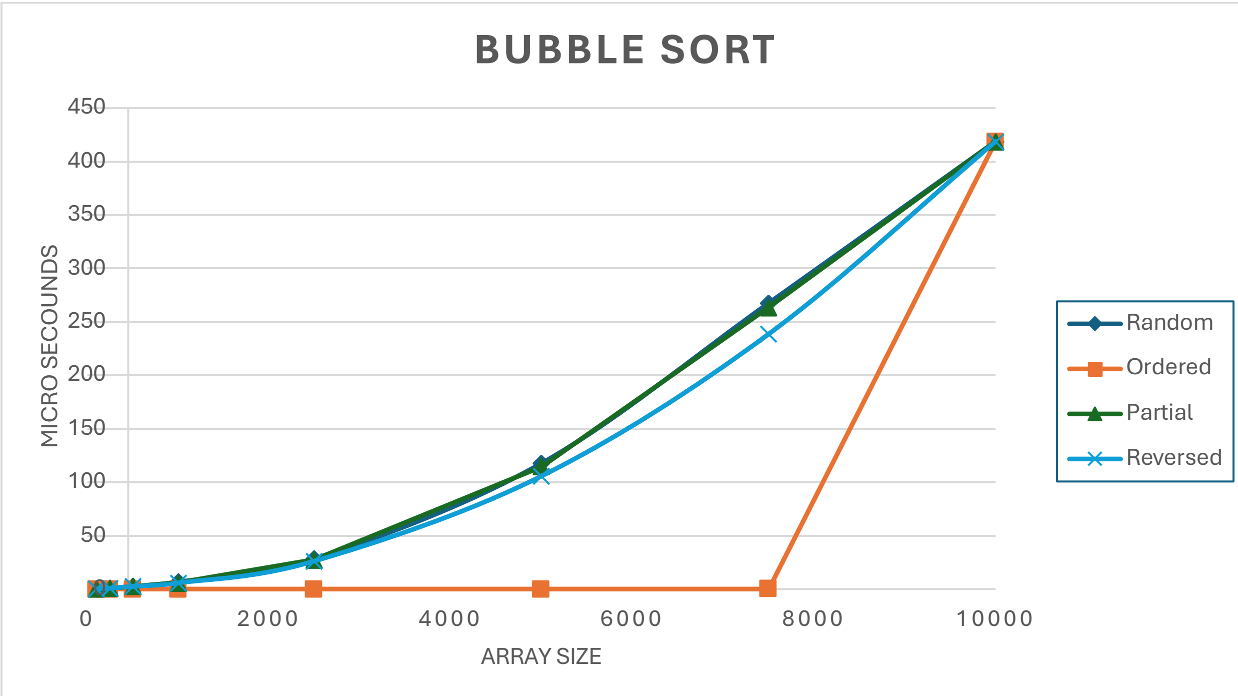
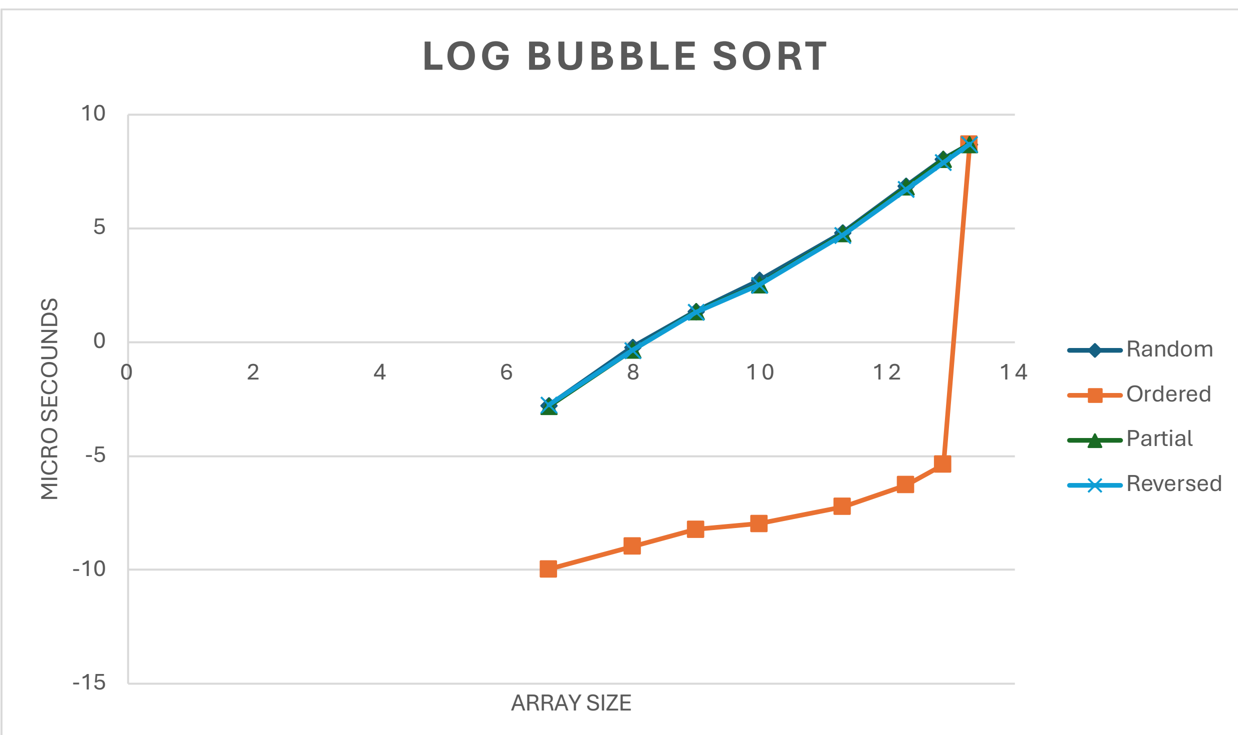
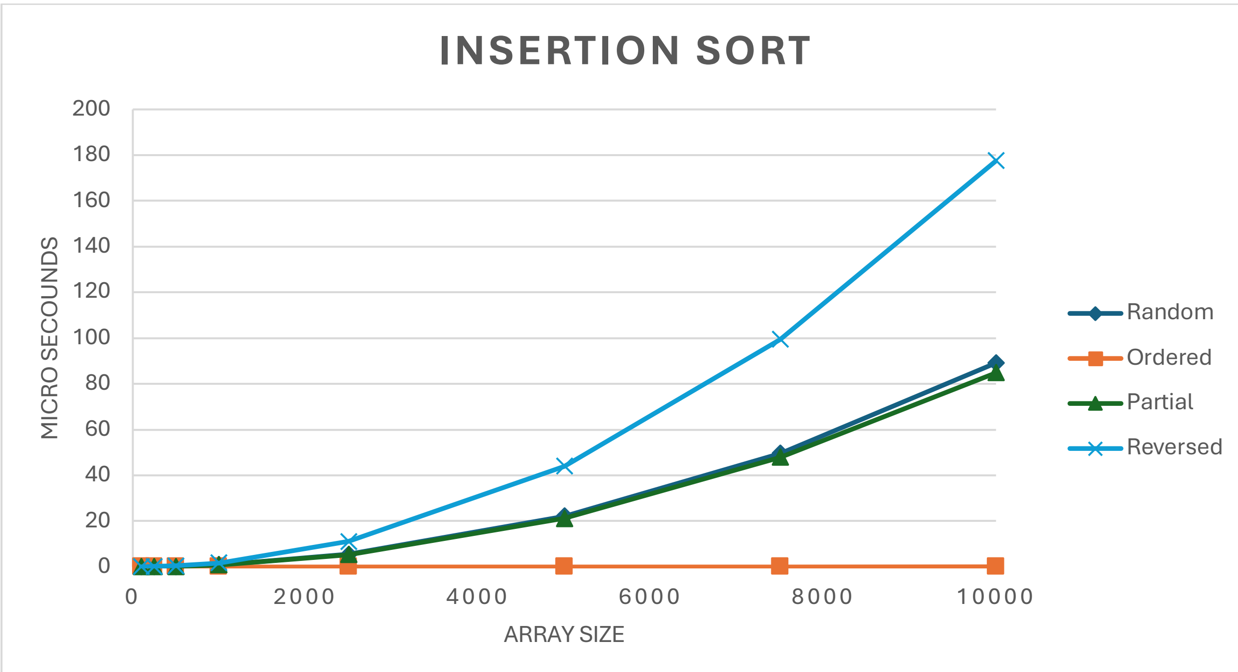
BUBBLE SORT

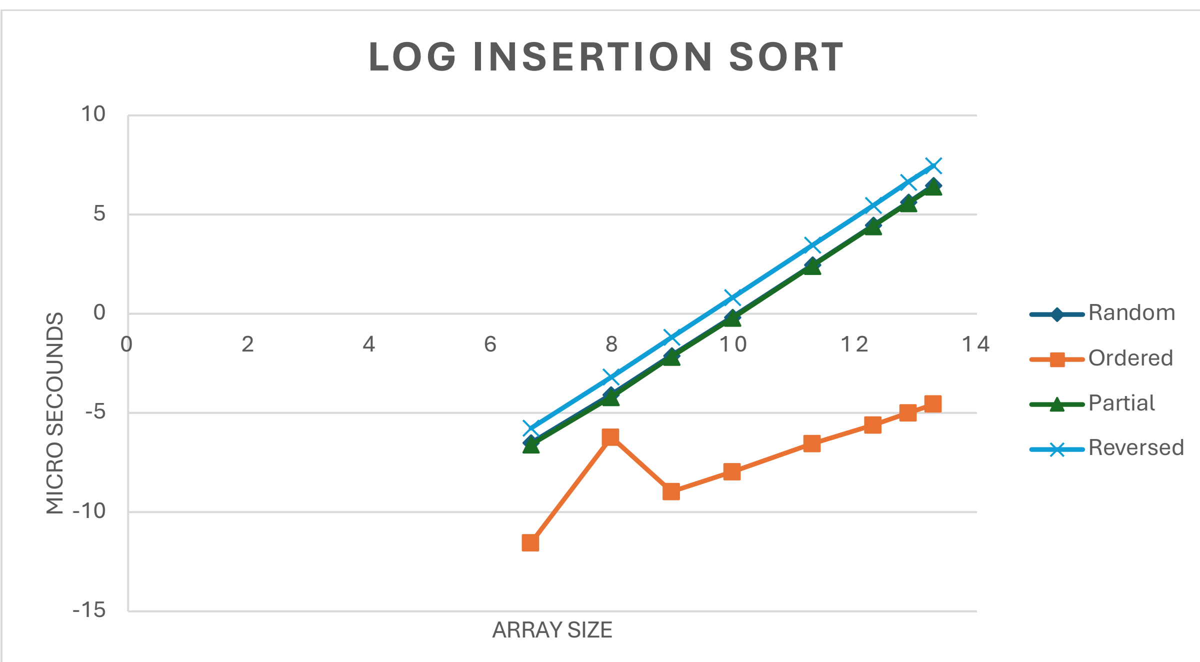

BUBBLE SORT
450
400
350
300
250
200
150
100
50
0
2000
4000
6000
8000
10000
MICRO SECOUNDS
ARRAY SIZE
Random
Ordered
Partial
Reversed


LOG BUBBLE SORT
10
5
0
-5
-10
-15
0
2
4
6
8
10
12
14
MICRO SECOUNDS
ARRAY SIZE
Random
Ordered
Partial
Reversed

The bubble sort graphs show that sorted algorithms are O(n), and the other lists show that the algorithm is O(n^2). This is expected from bubble sort; since, bubble sort is O(n^2) unless created a certain way to account for an already sorted list than it is O(n).

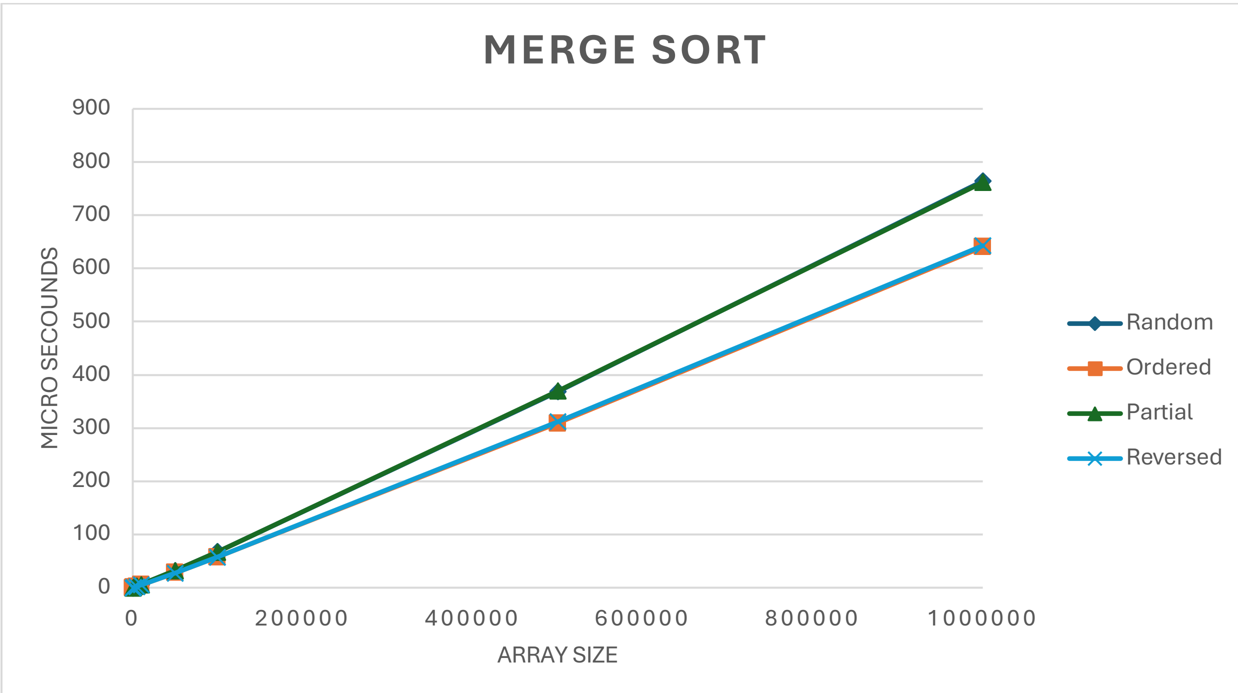
INSERTION SORT

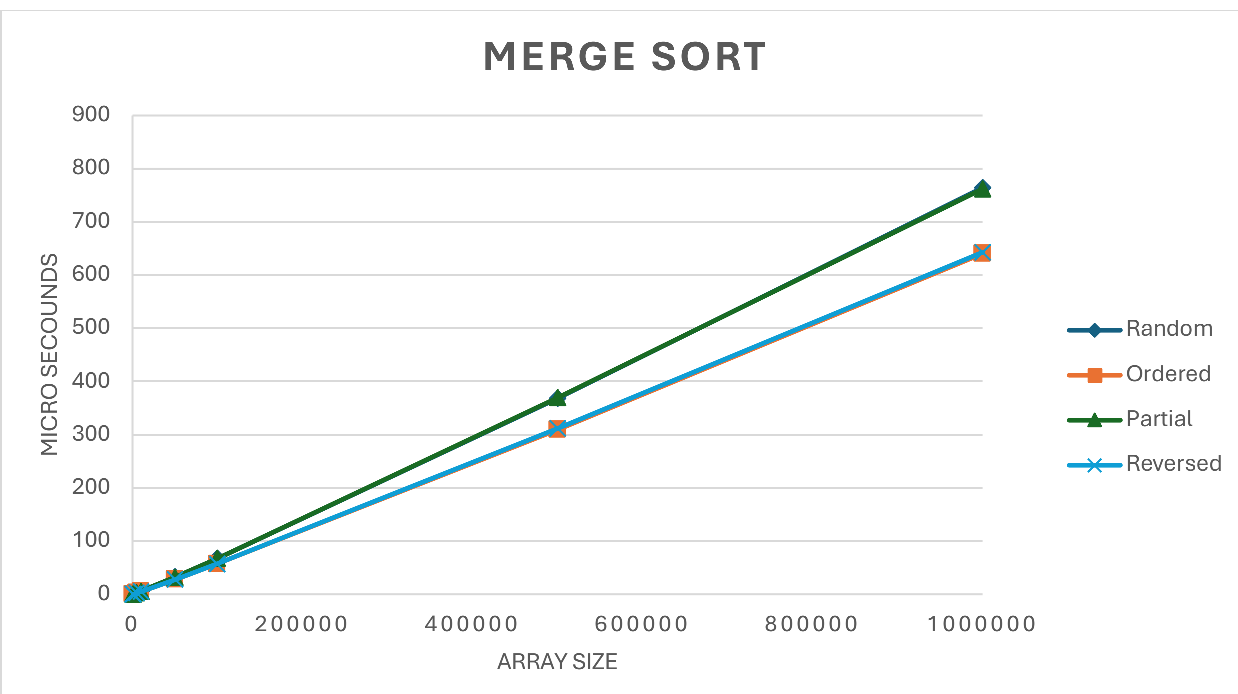




The graphs indicate that normally the algorithm is near O(n); however, when the list is reversed ordered the algorithm is O(n^2). This is expected from insertion sort; the best case is O(n), but the worst case (reverse order) is O(n^2).

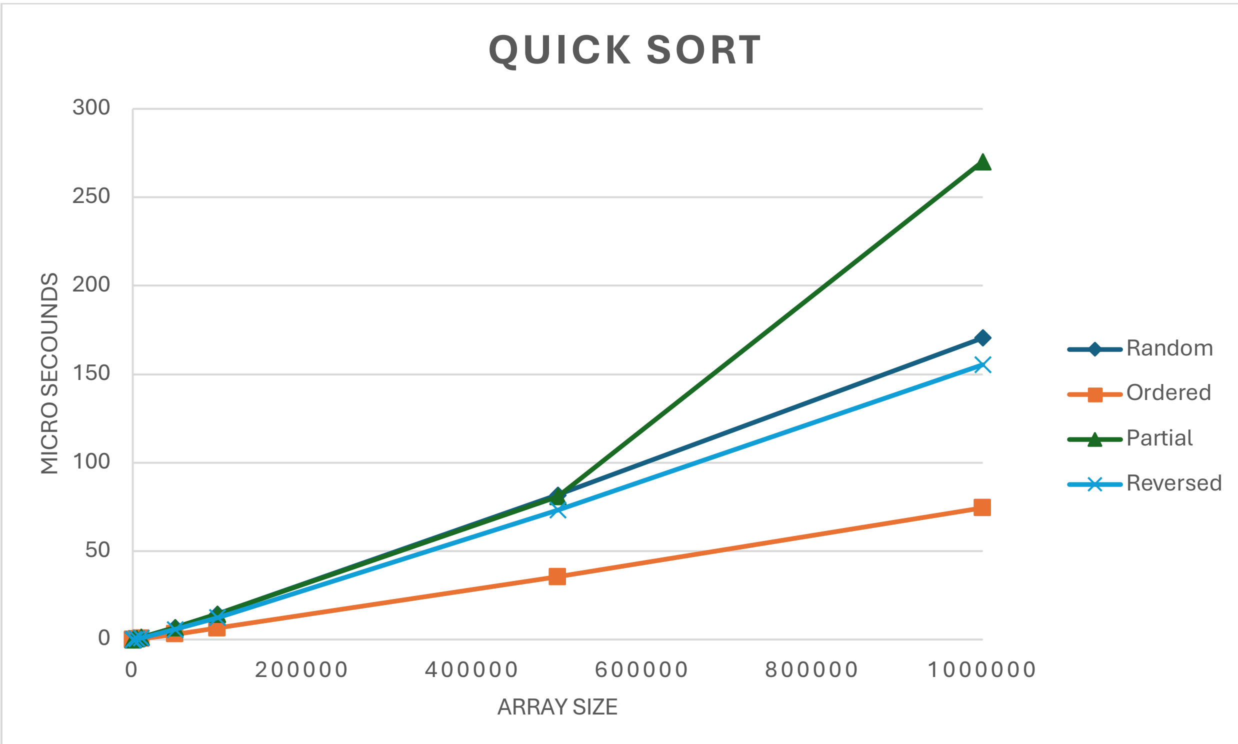
MERGE SORT

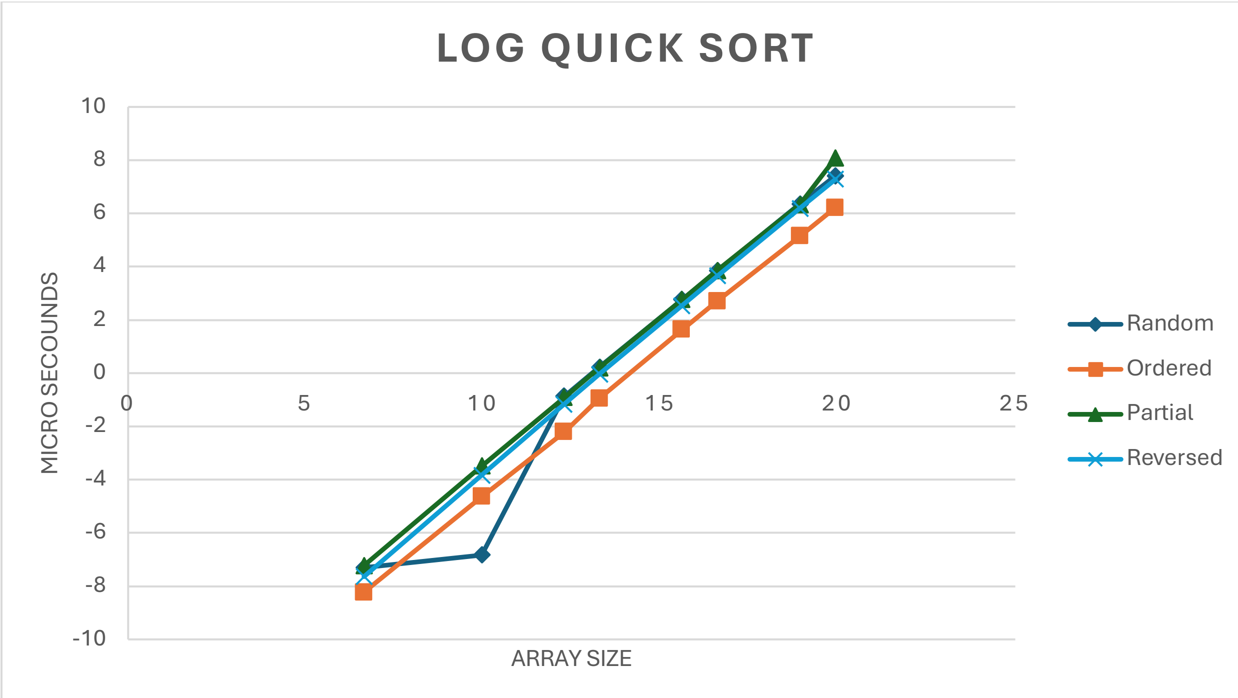




The graphs show that merge sort is O (n log n). This is expected from merge sort, since; merge sort is commonly O (n log n).

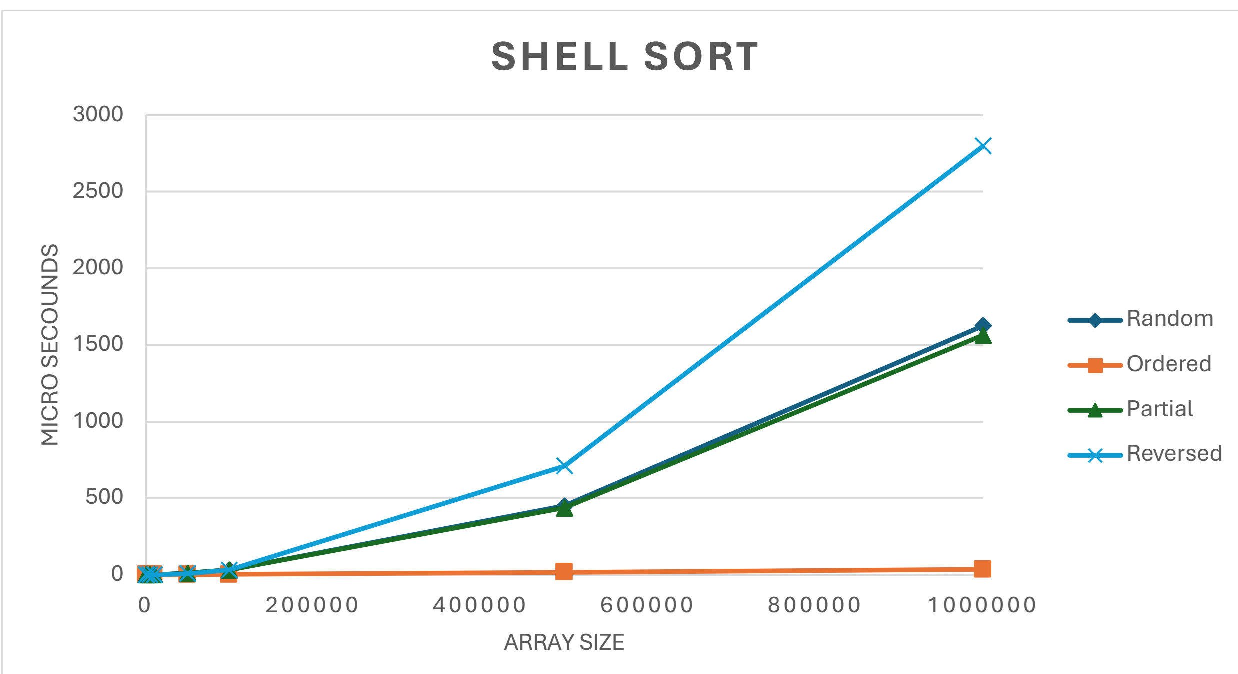
QUICK SORT

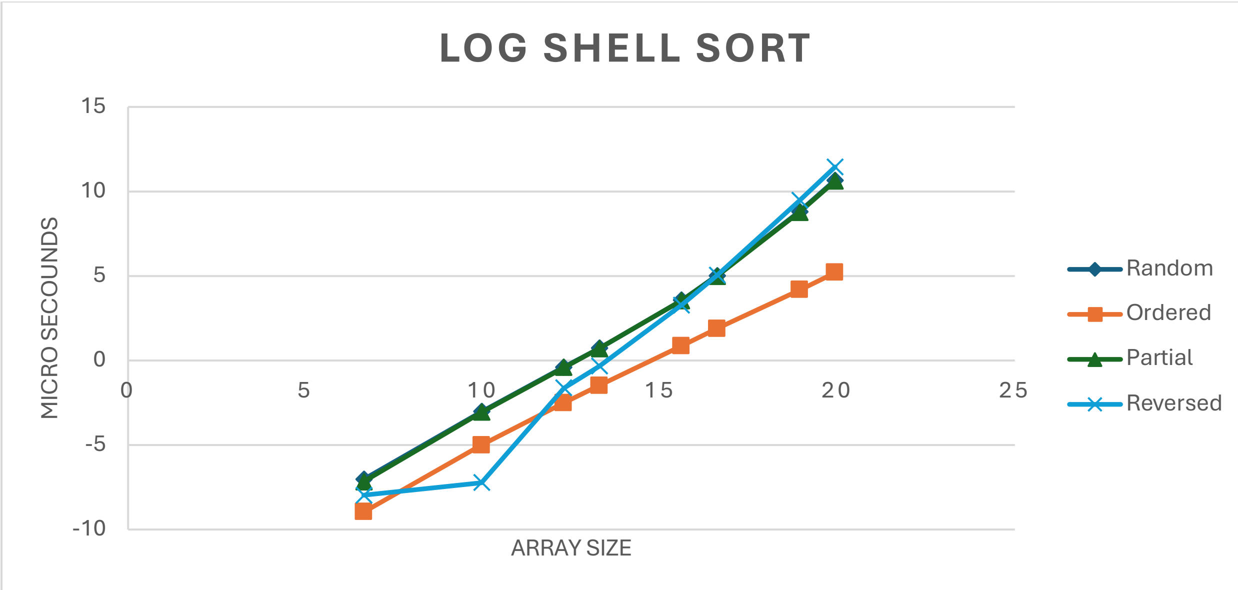




The graph is showing that quick sort is O (n log n). This was expected since I coded the partition to be a “median of three”; which mitigates the O (n^2).

SHELL SORT





The graph indicates that the ordered list is O (n log^2 n), and the rest of the lists are O (n^3/2). This was expected since shell sort is often O (n^3/2) except in certain conditions such as an ordered list where shell sort is O (n log^2 n).

CONCLUSION

- **Bubble Sort**: Best case O (n), worst case O (n^2) (exponent 1 and 2).
- **Insertion Sort**: Best case O (n), worst case O (n^2) (exponent 1 and 2).
- **Merge Sort**: Best case O (n log n) worst case O (n log n) (exponent around 1.5)
- **Quick Sort**: Best case O (n log n), worst case O (n^2) (exponent around 1.5 and 2).
- **Shell Sort**: Best case O (n log^2 n), worst case O (n^3/2) (exponent around 1.6 and 1.5).