|  |
| --- |
| BSHCE3 Computing (Part Time) – Advanced Programming |
| Compare Sorting Algorithms |
| Comparing the Bubble Sort and the Insertion Sort in terms of efficiency |

|  |
| --- |
| Jonathan Flanagan  X18143890 |

Table of Contents

[Testing Method 1](#_Toc55413664)

[Report: 2](#_Toc55413665)

[Comparisons 2](#_Toc55413666)

[Swaps 3](#_Toc55413667)

[Complexity 5](#_Toc55413668)

[Bubble Sort 5](#_Toc55413669)

[Insertion Sort 5](#_Toc55413670)

[Efficiency 6](#_Toc55413671)

[Conclusion 7](#_Toc55413672)

[Further Tables 7](#_Toc55413673)

# Testing Method

For testing the two algorithms, 3 lists where chosen where the number of elements doubled for each subsequent test. The first list comprised of 10 elements, second 20 elements, third 40 elements.

After testing at multiple sized lists, lists of these sizes where chosen for the report as graphing results to scale on larger data sets was not viable due to the variances growing greater with larger lists. (Displayed in chart boxes at the end of report, [Figure 6])

To make sure the efficiency was tested equally, an initial set of predetermined elements at random order (Human defined in code) was used as the basis for creating each list. Once each list was created it was tested against the bubble sort algorithm, its results recorded and then the same list used for the bubble sort was recreated and tested against the insertion sort algorithm. This was done to mitigate any variance in the lists and negate the possibility of any elements falling into the “correct” order when randomising with a shuffle.

Three secondary tests where also carried out, 1) list of ten elements in reverse order and sorted order to get best and worst case scenarios, 2) 4 lists in reverser order of size 10,20,30,40 to compare Insertion sort swaps and shifts to Bubble sort swaps. 3) Ten lists of size 10 with each list ranging from zero items sorted to all items sorted and number of comparisons were recorded.

# Report:

## Comparisons

In the tests with 10,20 and 40 pre-set random order elements the bubble sort always completed n(n-1)/2 comparisons on each list. [Figure 1]

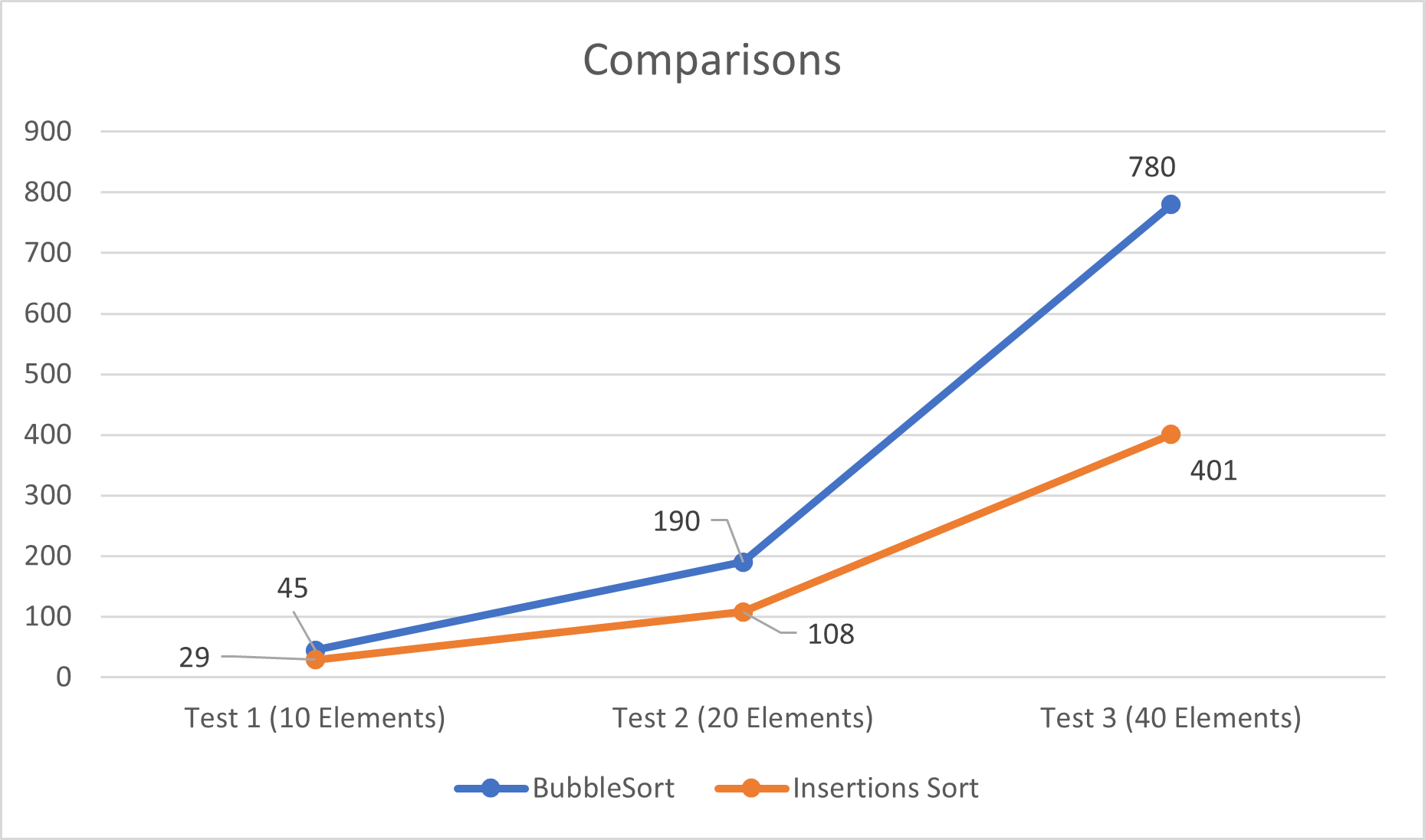
With a secondary test of two lists of 10 elements, one sorted and one in reverse order, the Bubble sort also always completed n(n-1)/2 comparisons. It can be concluded then that bubble sort best case and worst-case scenarios for comparisons is the same at n(n-1)/2. [Figure 2]

While for insertion sort it depended on the order of the list for how many comparisons where done. For the tests with 10,20 and 40 elements in the pre-set random order the number of comparisons completed by the Insertion sort was roughly close to n(n-1)/4. [Figure 1]

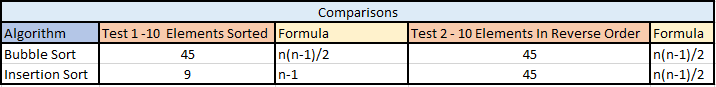
With the secondary test for Insertion sort on sorted and reverse order lists of 10 elements, the number of comparisons completed was n-1 (for the already fully sorted list) and n(n-1)/2 for the list where every element is in reverse order. [Figure 2]

With these results we can conclude for Insertion sort, best case scenario is n-1, worst case scenario is n(n-1)/2 and the average scenario is roughly n(n-1)/4 for comparisons. Therefore, the average amount of comparison completed by Insertion sort was roughly half of the comparisons completed by Bubble sort.

Figure



Figure



## Swaps

A swap as defined in Bubble sort is when two elements are swapped with each other. With Insertion sort a swap is defined as, when an element that was copied out of the list is inserted back into the list at its correct position.

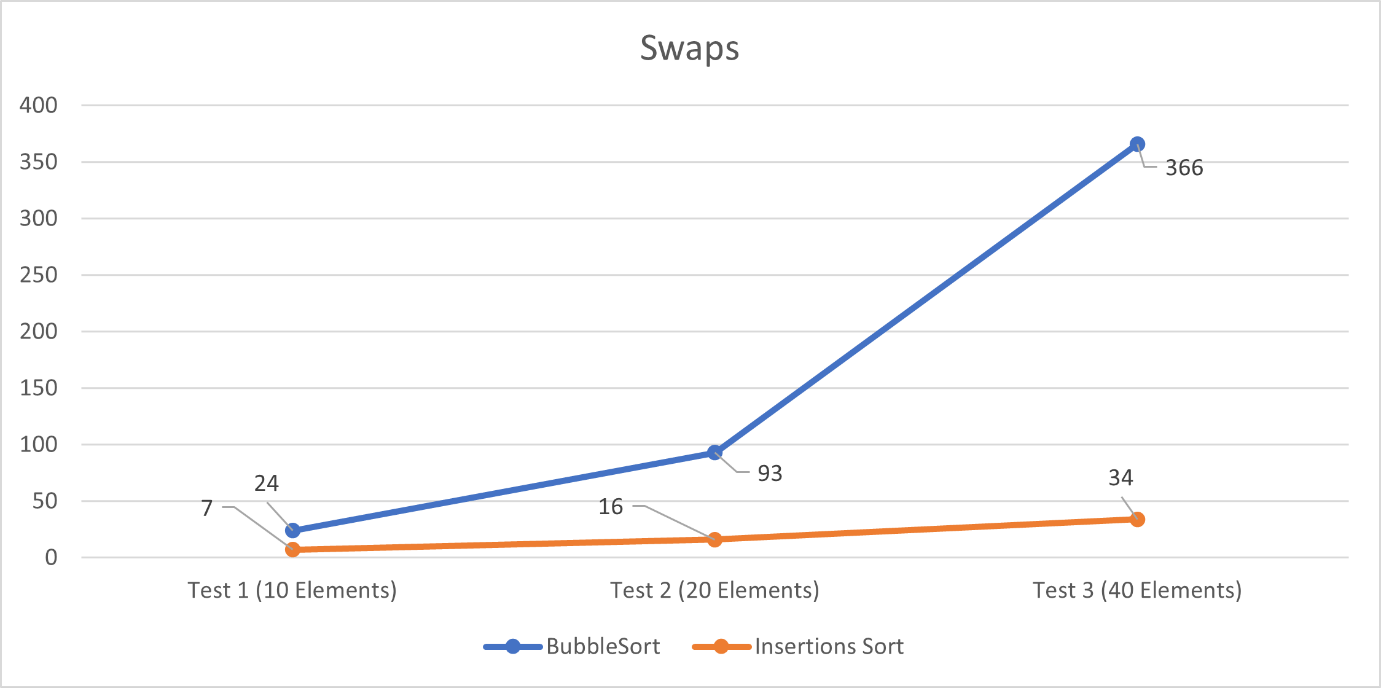
**For example**, an array of integers [2,3,4,5,1]. To sort in ascending order, for insertion sort: [1] is copied from the list, compared against each element, and inserted in its correct position at index 0 while all other elements where shifted along the array. This is counted as 1 swap, as only 1 element was removed and placed back into the array. For Bubble Sort, [5] is swapped with [1], then [4] is swapped with [1] then [3] is swapped with [1] and then [2] is swapped with [1]. In this scenario, 4 swaps have been completed.

Recording the swaps was done during the same tests as recording comparisons, with lists of size, 10, 20 and 40 with pre-set random order elements.

For these tests, the number of swaps performed by the Bubble sort was always greater than then number of swaps performed by the Insertion sort. Judging by the trend in the graphs [Figure 3], it is a fair conclusion to extrapolate that this will always be the case with a widening gap, for lists of random elements “almost” sorted.

It is worth noting that while the number of swaps carried out by Insertion sort was less then those performed by the Bubble Sort, the number of elements shifted in the Insertion sort was identical with the number of swaps in Bubble sort.

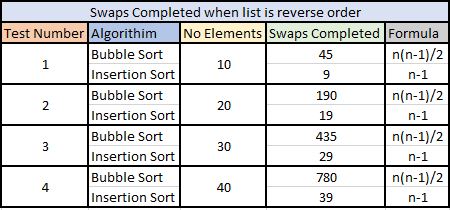
Figure



In a test where each element in the list is unique, such a list of integers where no 2 elements are the same integer, and the list is in reverse order. The Bubble sort always completed n(n-1)/2 swaps whereas the Insertion sort always completed n-1 swaps. [Figure 4]

This should **not** be confused with “shifts” carried out by the Insertion sort. As in this scenario although only n-1 elements are copied from and re-inserted to the list, every possible comparison and shift of elements is carried out with results of n(n-1)/2. Therefore, in this case, theoretically, Insertion sort runs no faster than Bubble sort.

Figure



We can conclude from these tests that for a random set of data the in terms of swaps completed, Insertion sort is more efficient than Bubble sort.

## Complexity

With the results above it is worth comparing their complexity. To calculate the complexity of the two algorithms we can look at their pseudo code and calculate the Big-O value.

### Bubble Sort

For i = 0 up to the number of elements **=> Complexity of n**

{

For j = 0 up to the number of elements -1-i **=> Complexity of n**

{

If the element at j > the element at j+1 **=> Complexity of 1**

{

Swap the element at j with the element at j+1

{

{

{

**O(n\*n\*1) = O(n^2)**

Therefore, Bubble sort has a worst-case complexity of **O(n^2).**

### Insertion Sort

For i = 1 up to i < the number of elements, i++ **=> Complexity of n**

{

keyElement = the Element at i **=> Complexity of 1**

pos = i **=> Complexity of 1**

while pos > 0 and the element at pos -1 > keyElement **=> Complexity of n**

{

The element at pos = the element at pos -1 **=> Complexity of 1**

pos = pos-1 **=> Complexity of 1**

}

The element at pos = keyElement **=> Complexity of 1**

}

**O(n \* 1 \* 1 \* n \* 1 \* 1 ) = O(n^2)**

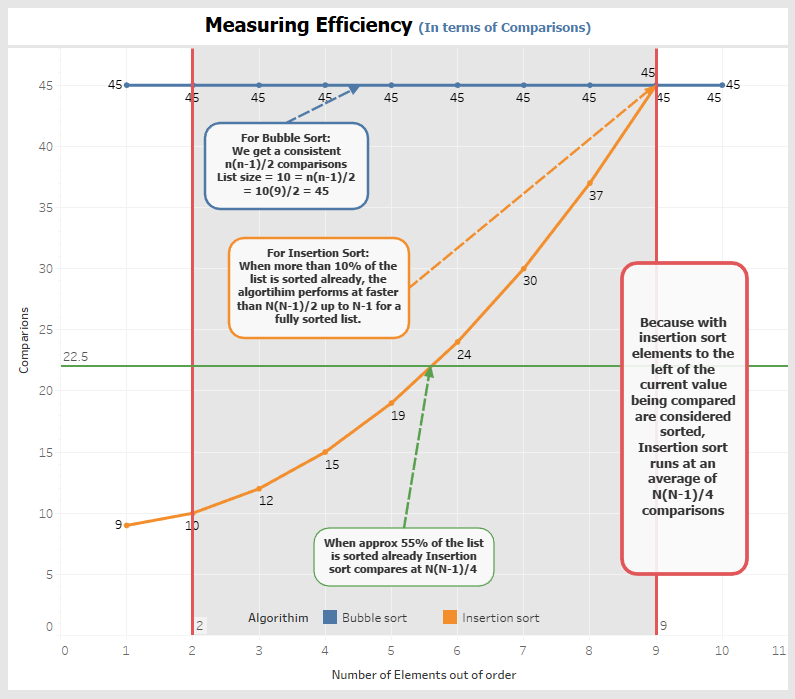
Therefore, Insertion sort has a worst-case complexity of **O(n^2).** With both algorithms having a worst case Big-O value of O(n^2), the difference is in the inner loops, as Bubble sort will always compare each element against the rest, Insertion sort will only enter its inner loop some of the time.

# Efficiency

To further test the efficiency a test was carried out on ten separate lists each containing 10 elements, for each list there was a specific number of elements out of order from 0 to All Elements. The results can be seen below. [Figure 5]

It can be seen in the graph below that when a list of elements is already at approx. 55% sorted then Insertion sort will compare at N(N-1)/4. Since Insertion sort assumes that elements to the left of the item being sorted are sorted already, Insertion sort will compare on average on each pass n(n-1)/4 elements before the correct position is found. That is inline with the previous examples where Insertion sort completed approximately half the amount of comparisons that Bubble sort completed.

It can also be seen from the graph that when starting to sort a list, if more than 10% of the list is sorted already then the number of comparisons carried out by Insertion sort will be less than N(N-1)/2 whereas Bubble Sort compares at a consistent rate of N(N-1)/2 no matter what % of the list is already sorted.



Figure

# Conclusion

With tests done between the swaps/comparisons of each algorithm, when a list of random elements is to be sorted, or a list of “almost” sorted elements needs to be sorted. Insertion sort will be more efficient than Bubble sort.

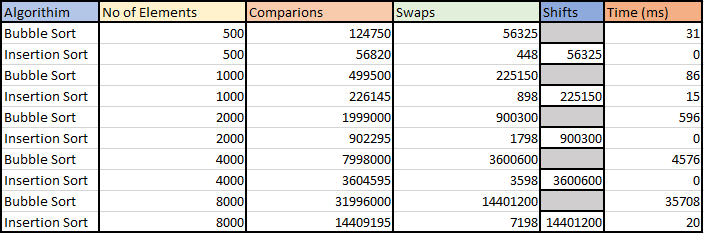
From the tests carried out, it can be reasonably concluded that the act of copying an element out of the list once, for reinsertion at its correct location, is less time consuming at execution time than the continuous swapping of two elements at their indices in Bubble sort [Figure 6]. Also when comparisons are completed by each algorithm the Bubble sort will always complete n(n-2)/2 comparisons while Insertion sort will vary between n-1 and n(n-1)/2 with an average comparison amount of n(n-1)/4.

If the list is in reverse order to the order in which it needs to be sorted in, then there is no theoretical difference in the performance of each.

Overall, three conclusions can be met:

1. In most real-world examples, where there is a list of random, almost sorted data, although both algorithms are a Big-O complexity of O(n^2), Insertion sort will be more efficient than Bubble sort in terms of execution time/comparisons/swaps.
2. When a list has approximately more than 10% of its items sorted already, Insertion Sort is more efficient that Bubble Sort in terms of comparisons.
3. Insertion sort will always be more efficient in terms of swaps.

# Further Tables



Figure