**Evaluation**

|  |  |  |
| --- | --- | --- |
| **Test** | **Insertion Sort** | **Quick Sort** |
| **1** | **73** | **83** |
| **2** | **30** | **108** |
| **3** | **2840** | **1008** |
| **4** | **363** | **1563** |

1. In 3 out of 4 test, insertion sort had performed better than quick sort.
2. On elements sorted in descending order quick sort had performed faster than insertion sort, despite it is the worst performance case for both algorithms (O(N2)). This can be seen at test 3.
3. Most of the elements in test4.txt are in ascending order, as seen from the poor result of quick sort algorithm (O(N2)) and O(N) performance of insertion sort algorithm.
4. In sum for all 4 tests quick sort has made less comparisons than insertion sort. 2762 by quick sort vs 3306 by insertion sort.
5. In test one the two algorithms have shown the smallest difference in comparisons. E.g. similar performance. (Difference of 10 comparisons).

|  |  |  |
| --- | --- | --- |
| **Test** | **Insertion Sort** | **New Sort** |
| **3** |  | **7388** |
| **4** |  | **4971** |
| **5** | **2728** | **1682** |

1. The new sorting algorithm (the NewSort) perform better than insertion sort algorithm in test 5 (test5.txt). This is due to two reasons: 1) Most of the numbers in test5.txt are in reverse order (which is the worst performance case for the insertion sort algorithm). 2) test5.txt contains duplicates, which favours performance of the NewSort algorithm.
2. NewSort algorithm’s best performance occurs when all the elements of the array are duplicates. O(N);
3. NewSort algorithm’s worst performance occurs when all the element of the array are unique. O(N2)