Activity 4: Exploring Run Times

|  |  |  |  |
| --- | --- | --- | --- |
| **Sorting Runtime** | | | |
| Sort Size | Sort Type | Sorting Order | Run Time (2 d.p.) |
| 1500 | Bubble Sort | Ascending | 27.67 |
| 1300 | Bubble Sort | Descending | 1496.67 |
| 800 | Bubble Sort | Random | 666.33 |
| 500 | Bubble Sort | 10% ordered | 364.67 |
| 1500 | Selection Sort | Ascending | 1613.33 |
| 1300 | Selection Sort | Descending | 1697 |
| 800 | Selection Sort | Random | 619.33 |
| 500 | Selection Sort | 10% ordered | 361.67 |
| 1500 | Insertion Sort | Ascending | 30 |
| 1300 | Insertion Sort | Descending | 1661 |
| 800 | Insertion Sort | Random | 244.33 |
| 500 | Insertion Sort | 10% ordered | 285.33 |

|  |  |
| --- | --- |
| **Average Sorting Runtime** | |
| Bubble Sort | 638.84 |
| Selection Sort | 1072.83 |
| Insertion Sort | 555.17 |

Bubble Sort: In terms of time complexity, it was most evident that when bubble sort had to sort in reverse order, it took a runtime of 1496.67 in comparison to the ascending direction with a runtime of 27.67. This shows that bubble sort is efficient when the data is already sorted in ascending order, but in reverse order, it will take much longer as it needs a lot of passes to swap it in the direction needed. Bubble sort takes the longest in descending as the sort is designed to move the largest value to the end, thus multiple passes are required to sort it in order.

Selection Sort: Selection sort is the slowest sort out of the 3 sorts investigated as it requires one full pass to find the smallest value within the data range. This sort is the most time consuming as each task can only find 1 smallest element at a given time, swapping after the pass finishes. For this particular sort, it is also known as an unstable sort, if given 8, 8, 3, the first 8 will swap with the 3, putting the first 8 after the 2nd 8, thus selection sort cannot guarantee stability of the array.

Insertion Sort: The average runtime of an insertion sort is found to be the quickest compared the other two sorts. This sort is relatively fast, but it is not simple to implement. It is very fast with almost sorted data as minimal swaps and comparisons are required. It is very slow on reverse-ordered data and lots of swaps are required to shuffle larger elements up.