Joshua Little

Professor Nick Murphy

CSE 2300

July 23, 2023

Insertion & Selection Sort Report – Programs/Specs

Machine specs -

GPU: Radeon RX 580

RAM: 16gb 3200mhz DDR4 CL16

CPU: Ryzen 5 5600

Storage: Crucial P5 Plus 2TB PCIe M.2

Operating System: Windows 10

Other programs running – (Parentheses are from Task Manager. Components/Processes)

Firefox (15)

Microsoft Word (7)

Discord (6)

Visual Studio Code (15)

Microsoft Excel (2)

Windows Explorer

Insertion & Selection Sort Report – Experiment, Results, & Explanation

The results of this experiment consistently demonstrated that Insertion Sort performs quicker than Selection Sort. Both insertion and selection sort use a complexity of O(n^2), so we can expect similar results. Although, performance may vary depending on the dataset and how sorted it is. Generally, insertion tends to be faster than selection in arrays that are at least slightly sorted. The reason for this is because insertion makes comparisons based on what is and is not already sorted, whereas selection will always compare every value.

To elaborate, insertion sort essentially creates a sorted part of the array and an unsorted part of the array. It is possible for parts of a randomly generated array to already be sorted, for example if an array generates “4322, 4564” as two random integers inside of it, and there are no values between them, then they are already sorted in the correct order and insertion does not need to sort them. Theoretically speaking, if a randomly generated array happens to generate completely sorted, then insertion sort can technically have a complexity of O(n) instead of O(n^2); this is highly unlikely in practice though. However, on the opposite end, if an array is completely unsorted, like being the reverse of the algorithm, then it has a complexity of O(n^2) and is not more efficient than selection sort. Because insertion sort can skip sorted values, it ends up being more efficient on average for any semi-sorted array than selection sort.

Selection sort on the other hand, has a few use cases where it may outperform insertion sort. Selection sort will always compare every value in an array. This means selection sort will always have a complexity of O(n^2), even in theoretical best-case scenarios. In contrast to insertion sort, selection sort will always have zero numbers in its sorted array when starting out and will always have to move every value into it.

Overall, insertion sort tends to perform faster than selection sort despite sharing the same complexity of O(n^2). The results of this experiment concur with the theory that insertion is generally faster. Selection sort may be able to pull ahead when working with small datasets or highly organized data. Insertion sort tends to perform better in practically every other scenario. This experiment’s findings agree with the idea that insertion should be used for large and unorganized datasets, and selection sort should be used for small datasets and organized ones.

Observations –

This project taught me the situations in which I should use insertion or selection sort based on the dataset; that even if sorting algorithms share the same complexity some may be faster than others depending on situation, and that certain languages are more efficient at certain tasks than others. In discussion with other students and with the professor, I learned that the individuals doing this experiment in non-Python languages had a much more efficient time with this experiment than I did, using Python. It was interesting to learn why certain algorithms can be more efficient than others and will certainly be useful if I ever must program an algorithm like this one. Overall, I think my main takeaway is to use the right programming language and the right algorithm for the right job to achieve maximum efficiency, and also understanding of how to figure out which one is the right one to use.