**HW Review #4 CS – 2336.003 Andy Nguyen | adn200004**

1. **Compare and contrast the three O(n2) algorithms with each other**

For selection sort, assembling a sorted array requires the algorithm to repeatedly find the smallest value of the unsorted part of the array and swapping it to the front index.

For bubble sort, assembling a sorted array requires the algorithm to repeatedly compare adjacent elements and swap them into appropriate positions.

For insertion sort, assembly of the sorted array is determined by having sorted and unsorted parts of the array. An element is chosen from the unsorted part of the array and put in its correct order in the sorted array.

* **Swaps Analysis**

Starting with the bubble sort and insertion sort, I expected the number of swaps between the sorts to be the similar because both sorts move each element to its appropriate position in the array through a series of swaps in relation to their adjacent elements. Looking at the output of the program, I saw that the number of swaps were the exactly same demonstrating their similar inner loop methods. Selection sort I expected to have the least number of swaps due to the inner loop of the sort finding the smallest value in the unsorted part of the array and swapping it to the front index. Selection sort had the least swaps due to its unique inner loop method requiring less iterations.

* **Comparison Analysis**

For insertion sort, I believe the number of comparisons is going to be substantially smaller compared to the other **O(n2) algorithms**. The reason I believe so is because if the inner loop comparison does not require a swap, we don’t have to traverse the rest of the sorted part of the array cause its already in the right spot. Upon seeing the results, my hypothesis was accurately represented with the number of comparisons being half of the number compared to selection and bubble. Like when sorting a deck of cards, we insert a card in its appropriate position without having to check all other cards. For the number of comparisons regarding bubble and selection sort, I believe the number of comparisons to be the same because their inner and outer loop structure is essentially the same. Looking at the results, my guess was correct and that having the same loop structure and being in the same **O(n2)** time complexity resulted in the same number of comparisons.

1. **Compare and contrast the two O(log n) algorithms with each other**

For quick sort, a divide and conquer algorithm, picks a pivot element, and split the array into partitions based around that element. Elements greater than the pivot move to the right partition and elements less than move to the left. The process is repeated, and the partitions are combined once sorted.

Another divide and conquer algorithm, merge sort firstly divides the arrays into halves until it can no longer be divided ultimately leaving one element in a list. Proceed to sort the sub lists and recursively combine them back together forming the sorted array.

* **Comparison Analysis**

In terms of who will have the more comparisons, I believe merge sort to have less comparisons since at the third step of merge sort, having to halves of a list and our goal being to merge them, we only need at most the combined size of those two halves comparisons. Based on the results, my guess was accurately represented in the results. It could be further proven that the merge would have less comparisons than quick sort due to unideal pivot location resulting in more comparisons.

1. **Compare and contrast insertion sort with both O(log n) algorithms**

* **Swaps Analysis**

Based on the large data set of 5000 elements, I believe quick sort is going to have less comparisons than insertion sort because the Big O of log n has a shorter run time compared to n^2. With the results matching my guess, divide and conquer algorithms can eliminate unnecessary comparisons when dealing with the large data set.

* **Comparison Analysis**

Being that the Big O notation of insertion is a bigger run time compared to the O(log n) algorithms, it only makes sense that the insertion sort will have a lot more comparisons compared to quick and merge. To explain the greater comparison with quick sort against merge, we once again can refer to merge sort requiring only the number of comparisons of the combined size of the halved arrays and that a bad pivot element, specifically a max/min element being chosen, can substantially increase the number of comparisons.

**Overall**

Overall, we can especially see the difference in run time based on Big O notation of the sorting algorithm when dealing with large dataset. Selection, bubble, and insertion sorts, sorts which don’t utilize divide in conquer, have substantially larger comparisons due to their quadratic nature of O(n2). In terms of swaps, we see that selection sort comes out with the least number of swaps. It may seem confusing as to why selection has less swaps compared to the O(log n) algorithms, but the large number of comparisons required for large data set explains its O(n2) time complexity. Other than selection sort’s swaps, we can see from the results that the time complexities of the sorts are reflected in the number of comparisons and swaps.