Meghna Raswan

For this assignment, I used bubble sort, selection sort, insertion sort, quick sort, and merge sort to compare and contrast each of the sorting algorithm's functionality and their differences in runtime. Using all of these different algorithms, I compared their elapsed times in running the programs by converting the start and end time to milliseconds so I could get a better sense of the duration of each sorting algorithm. By calculating the time, I was able to figure out which sorting algorithm might have taken longer than another, depending on their elapsed times. From my outputs, I have come to the conclusion that merge sort was one of the faster sorting algorithms compared to the rest of the algorithms with a larger array of elements. Bubble sort, insertion sort, and selection sort were also relatively fast when given a large array.

Although bubble sort has the ability to repeatedly compare each adjacent item with each other, swapping the elements accordingly, with a time complexity of O(N^2) and space complexity of O(1), one of its disadvantages is that it is a relatively slow algorithm. Unless the comparisons in sorting is not accomplished with the first run through of comparisons, it will repeatedly compare each value until it is sorted from start to finish, especially if the array is large, amounting to a longer runtime.

Selection sort performs in-place sorting. However, since it has an O(N^2) complexity, just like bubble sort, it works efficiently with larger arrays. For smaller data sets, it is sufficient enough to sort quickly as an in place sorting algorithm.

Insertion sort can also perform simple comparisons with arrays as it can insert every array element into its proper position, sorting from subarrays to the overall array. Although the number of swaps is reduced than a bubble sort, follows a simple implementation, and can perform in=place sorting with a constant amount of O(1) additional memory, insertion sort, like selection and bubble sort, is more efficient when it comes to arrays containing lesser elements, as it also has an O(N^2) complexity.

Quick sort, one of the faster algorithms, is an internal algorithm that divides an array into 2 subarrays, and repeatedly divides these subarrays until they can no longer be divided, uses a pivot point to partition the elements, and then sorts accordingly by using the left partition to contain elements smaller that the pivot and right partition to contain elements greater than the key element. Though quicksort is significantly faster than other algorithms because its inner loop can be efficiently implemented on most architectures, the space complexity is O(logN) and the time complexity is O(N^2).

Like quick sort, merge sort splits into 2 subarrays continually until there is one element left. It uses three arrays where two are used for storing each half, and the third external one is used to store the final sorted list by merging other two arrays, sorted recursively. It has a worst case and average case that has the same complexities of O(NlogN). Although merge sort can work more efficiently and faster with data sets irrespective of their size, as opposed to quick sort, merge sort requires additional memory space to store the auxiliary arrays since it does not perform in-place sorting.

Meghna Raswan

I used C++ to conduct this empirical analysis and I think the algorithms have worked more efficiently ince C++ is a compiler, not an interpreter, performing faster, and used pointers for the elements in the array. Some of the disadvantages to conducting an empirical analysis is the accuracy and efficiency of how you program and run each sorting algorithm. Another downside is accurately comparing with an array you have built because some sorting algorithms work more efficiently with smaller arrays than others, making it seem like it has a better performance than another function. Overall, though many of these sorting algorithms have their advantages, working in some areas better than others, they still obtain many disadvantages as well.