**Sorting Algorithms**

Steven Boxer 11/22/2017

Sanford Leach

**Total Number of Comparisons:**

**Ordered** **Random** **Reversed**

**Insertion Sort: 9999 24,765,442 50,004,999**

**Merge Sort: 69,008 120,404 64,608**

**Quick Sort: 49,995,000 158,078 49,995,000**

**Top Sorting Algorithm:**

Choosing the top sorting algorithm meant finding out which one has the quickest average time complexity while looking into space complexity. This was easy to eliminate Insertion Sort as the average case is O(N^2) because not only does the list have to be traversed, but also the element has to traverse through the new sorted list being created. Because both Quick Sort and Merge Sort have an average time complexity of O (N log2 N), deciding which one was the best method came down to it’s worst case scenario. Merge Sort will always be O (N log2 N) because the algorithm will always need to traverse the array N times and do this for a dividing “Log2 N stages” thus keeping the algorithm at a constant time complexity that, for significantly larger arrays, makes a dramatic difference. The only downside to this method comes if space is a critical factor and the array dealt with is large. Unlike Merge Sort, Quick Sort has a worst case time complexity of O(N^2). This stems from the situation where the array is already sorted, where the split point would cause very skewed splits with one array containing the first split point and everything after in the second array. This can also be seen through the total number of comparisons. For random, which we presumed as the average case, you can see that the number of comparisons vary drastically from Insertion compared to the other two methods. Because of this, the optimal sorting algorithm to use with no external factors is the Merge Sort. Stated at the beginning though, space can be an issue. If this is the case, then quick sort may be the better sorting method. This stems from the idea that you are constantly creating new arrays or vectors for merge sort, putting it at a worst case O(n) space complexity. On the other hand, quick sort remains at a worst case O(log n) space complexity meaning, if space is a factor, then quick sort may be the method of choice.

**Extra Memory Space:**

We used Vectors for our sorting methods instead of arrays. This is because when we were dealing with pointers, we struggled with iterations. We were able to create blocks of memory on the spot. This also helped for our extra memory space which we needed for merge sort. Extra memory space was needed because of the way merge sort works, allowing us to make vectors on the spot. These vectors decreased in size as the recursive function was continuously called as the previous called vectors were split.