Joseph Morales

Algorithms and their strengths and weaknesses

To begin, each of the sorting algorithms have their own particular way of being more efficient than other algorithms. Starting with Shell sort it could have much better performance than other sorting algorithms like the Bubble sort or Insertion sort since it does not perform quadratics. The shell sort algorithm can be much more efficient but only on lower sized data sets. Shell will perform best on these sets since it is finding a pivot which could be any of the numbers in the sequence and dividing it from that position. This could mean with much larger data sets could choose a number far out of reach and be in a sequence of sorting an uneven portion. The algorithm is also more complex than other sorting algorithms with the implementation of it. For lower number data sets this algorithm works more efficiently. The data when using this sorting method did perform much better than most of the other algorithms but had larger spikes when the data set seemed complex. It did perform one of the fastest even compared to quick sort.

With using bubble sort it may be more comprehensible then some more complex algorithms. The bubble sort algorithm is also one of the few algorithms that does not need to create any temporary storage for placement of a number. The algorithm will sort all in the same array and when implemented it will be placed easily in the code. The downsides of using this algorithm can show when using larger data sets. The algorithm takes much longer to compute due to its single stage process. The bubble sort is not used in many larger scale real-life applications. It is much better for showing another way of sorting. When testing this algorithm it was much slower than any other algorithm. With data sets and complex numbers this algorithm would not be the main one used. But the implementation of the algorithm was easier than most.

Insertion sort can be much more efficient than bubble sorts implementation. Insertion sort is very simple to understand and implement into a data set. When using insertion sort it's ideal to use it for small data sets. It is also an in-place sorting algorithm which means when using the sorter it will not need to create an extra space for the input data and will use the same “in-place” data it has stored. Although the sorting algorithm works well with smaller sets when given high number sets of data it significantly drops its speed of data and can not compete with other algorithms. The insertion sort had a very smooth chart when it came to sorting. The algorithm compared to other slow algorithms had much better results and more consistency.

Selection sorts are very similar to insertion sort with all the same aspects like quick data processing on lower number data and being an in-place data algorithm but this sort will perform like a bubble sort in terms of time. With selection sort the sorting algorithm will take a leap when given datasets with higher numbers. This sorting method also does not keep an order of the elements when computing. Selection sort was another long sorting algorithm when it came to large sets. The algorithm was faster than the insertion sort or bubble sort and had the smoothest gradual graph compared to any sorting algorithm.

The quick sort algorithm can be one of the fastest and efficient algorithms compared to the rest. It is able to compute a vast amount of data compared to the other sorting methods. Quick sort also sorts in place so it does not need to store any values making it save less memory and space. This sorting algorithm seems the most ideal but can fall into trouble when facing datasets with a complex amount of numbers. This can cause the algorithm to perform like other algorithms with slower times. When testing the algorithm it had a very fast speed compared to algorithms with similar characteristics. It did seem more ideal when it came to large data sets since it would recover as the graph grew.

When using the heap sort algorithm it could be used for any number of data sets. This algorithm is one of the most versatile algorithms compared to the rest. This algorithm is also an in-place sorting algorithm which could make this method use less memory. The heap sort does need much more space than other algorithms since it requires the sorting of a tree system instead of other array types. In my data I've noticed the data had a more consistent chart then the other faster algorithms. But was at a more steady incline than others.

Lastly is the merge sorting method which can be used for many different applications since it has a very flexible scalability. When using this sorting method it could be more complex to implement into code but can be more reliable and stable since it holds the data and input. The merge sorting can fall behind since it has to store the data and hold more memory when performing and since the merge is the same all the time the data performance is based. When testing the algorithm the data showed much more peaks and valleys then other fast algorithms. It did perform faster than many but the data was not consistent making this method not more reliable for time.