Erik Davis
CS310
Assn #7 Analysis

Sorting Algorithm Analysis

Of the sorting algorithms we have studied, I decided to use my program to test the run times of the Selection Sort, the Shell Sort, and the Quick Sort algorithms.

The Selection Sort algorithm has a best, average, and worst Big O notation of $O(n^2)$ for the amount of comparisons it needs to perform. This make it the least efficient sorting algorithms among the ones we have learned, and it certainly reflected this in my test results. The Selection Sort had an average processing time of 1738ms, which is several times more than the other two algorithms tested.

The Shell Sort algorithm performed much better than the Selection Sort, with having an average processing time of 13ms. This is a huge improvement over the $O(n^2)$ algorithms. The Shell Sort algorithm (although current research and debate are still being carried out) has a best efficiency of $O(n^{7/6})$, an average of $O(n^{5/4})$, and a worst of $O(n^2)$. The fact that this algorithm performed almost as well as the Quick Sort may be attributed to how the gap was determined. While the regular $n/2$ method was used for the gap, it may have worked well with the even-sized list of almost entirely random numbers that was used.

Finally, we come to the Quick Sort. Always well known for its speed in sorting large lists of elements known to be out of order, the Quick Sort has a best and average efficiency of $O(n \log n)$ and a worst efficiency of $O(n^2)$. In the tests, the Quick Sort performed the best with an average processing time of 9ms (which I was astonished when I realized it could sort 50,000 elements in that short amount of time). However, the success of this algorithm can be attributed to a few controlled factors. The first is that the randomness of the list was controlled, and it was incredibly unlikely that the list would have any kind of order, especially since duplicated were allowed. Secondly, I did implement the method of choosing the median pivot point from three elements from the list, instead of just selecting the first element. I believe these two things together really helped the algorithm push toward it best efficiency of $O(n \log n)$.

The results reflect what I expected from the Big O efficiency notations of the algorithms, although I did enjoy implementing the tests myself and seeing how they performed.