Runtime Analysis

Selection Sort

Big-O: n^2

Rough Running Time: The Selection sort will walk through the algorithm from the current pass element (starts at index 0) all the way to the final element, locating the minimum element. The smallest element is then swapped with the current pass element. This continues until the current pass element is the last element. The main runtime comes from having to walk through the array n times, where n-1 is equal to the total size of the array. This leads to an average complexity of N * N/2, which equals N^2 in Big-O.

Gold's Pore Sort

Big-O: n^2

Rough Running Time: The Gold's Pore Sort consists of 2 half passes: one even pass and one odd pass. The array is walked through comparing even pairs in the first half pass, followed by odd pairs in the second pass. It is initially assumed by the program that the array is sorted; however, if a swap is needed, the array will be flagged as not sorted, requiring another pass. The rough running time is rather large due to the Gold's Pore Sort having lackluster run time speed, due to having to essentially make 2 passes each pass until the algo has been fully sorted. This leads to a complexity of (N-1) * (N-1), which results in a Big-O of N^2.

Quick Sort

Big-O: N*Log(N)

Rough Running Time: The Quick Sort begins by selecting a pivot element (we used the farthest right elem; however, that is not required). Once selected, every element is compared to the pivot element, moving smaller elements to the LHS and larger elements to the RHS. the elements in the left hand side and right hand side still need sorting, but the pivot is in the proper position. The largest element from the LHS is selected as the new pivot and the process repeats on the sub lists created from the LHS and RHS. The run time of the quick sort comes from recursively creating new lists and sorting these lists until you reach a single element which is in the proper position. It is a divide and conquer algorithm which still sees usage to this day. It averages a runtime of N*Log(N).

Merge Sort

Big-O: N*Log(N)

Rough Running Time: The Merge Sort breaks the array down into a bunch of singletons. Each Singleton is then compared with the one next to it, with the smaller one becoming the first element in a list now of size two. The lists of size two are compared with one another selecting the order of the list based on comparing the first index of each of the size two lists, placing the smaller elem into the new list of size 4. This continues until a single, sorted list has been created. Like the Quick Sort, it is a divide and conquer based sorting algorithm, breaking the list down into singleton's and then building it back up with each pass. It averages a runtime of N*Log(N), similar to the Quick Sort.