**Assignment 3 – Megan Bender**

**Run Time’s**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **10** | **100** | **1 mil Almost Sorted** | **1 mil Unsorted** | **10 mil** |
| **Bubble Sort** | <1 sec | <1 sec | 24 mins and 48.972 | 48 mins and 52.450 secs | n/a |
| **Selection Sort** | <1 sec | <1 sec | 21 mins and 12.365 secs | 21 mins and 2.669 secs | n/a |
| **Insertion Sort** | <1 sec | <1 sec | 39.129 secs | 14 mins and 49.608 secs | n/a |
| **Shell Sort** | <1 sec | <1 sec | 0.151 sec | 1.352 secs | n/a |
| **Merge Sort** | <1 sec | <1 sec | 0.445 secs | .92 secs | 2.452 secs |
| **Quick Sort** | <.1 sec | <.1 sec | 0.157 secs | 0.178 secs | 2.035 secs |
| **Heap Sort** | <.1 sec | <.1 sec | 0.203 secs | 0.273 secs | 4.214 secs |
| **Radix/Bin Sort** | <.1 sec | <.1 sec | 0.309 secs | 0.325 secs | 5.033 secs |

**Algorithm Runtime Classification**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Case** | **Average Case** | **Worst Case** |
| **Bubble Sort** | O(n2) | O(n2) | O(n2) |
| **Selection Sort** | O(n2) | O(n2) | O(n2) |
| **Insertion Sort** | O(n) | O(n2) | O(n2) |
| **Shell Sort** | O(n) | O(nlogn) | O(nlogn) |
| **Merge Sort** | O(nlogn) | O(nlogn) | O(nlogn) |
| **Quick Sort** | O(nlogn) | O(nlogn) | O(n2) |
| **Heap Sort** | O(nlogn) | O(nlogn) | O(nlogn) |
| **Radix/Bin Sort** | O(n\*k) | O(n\*k) | O(n\*k) |

**Justification for Run Time Classification**

Heap Sort: When looking back at our table we can see that heap sort runs slower than quick sort in every case. This happens because we are running two nlogn loops, from our walk up and walk down portions of the sorting algorithm. However, the advantage to using this method versus something like quick sort is the fac that we utilize much less memory.

Radix/Bin Sort: I did have to look more into this sorting algorithm because from my numbers in the above table it seems as though this sorting algorithm is one of the slower ones. That is however, only referring to the merge sort and on, and only taking into account the largest file has only up to 10 million values. The time complexity again is O(n\*k) due to the fact that we find the number of digits (k) within the largest number of and then loop through the binning and un-binning process that many times. This would take into account why it would take longer to run through more values versus and nlogn algorithm.