Gabriel Quiñones-Sanchez

CS 260

Project 3 Report

|  |  |  |  |
| --- | --- | --- | --- |
| Sort | Input Size | Execution Time | Growing Factor |
| Insertion Sort with an array | 5,000 | 244 milliseconds |  |
|  | 10,000 | 424 milliseconds | 1.74 |
|  | 100,000 | 17460 milliseconds | 41.18 |
| Insertion Sort with nodes | 5,000 | 526 milliseconds |  |
|  | 10,000 | 1811 milliseconds | 3.44 |
|  | 100,000 | 176490 milliseconds |  |
| Merge Sort with an array | 5,000 | 143 milliseconds |  |
|  | 10,000 | 244 milliseconds. | 1.71 |
|  | 100,000 | 3533 milliseconds | 14.48 |
| Merge Sort with nodes | 5,000 | 418 milliseconds |  |
|  | 10,000 | 996 milliseconds | 2.38 |
|  | 100,000 | 1631 milliseconds | 1.64 |
| Heap sort with array | 5,000 | 142 milliseconds |  |
|  | 10,000 | 211 milliseconds | 1.49 |
|  | 100,000 | 3350 milliseconds | 15.88 |

REPORT:

The big-0 for the insertion sort at best is O(n) and at worst O(n2). This is reflected in the data as seen by how fact the run time increases with the increase in input size. As it is doubled it increases slowly but surely and then by the time cases like 100,000 are tested the run time increases quadratically. The nodes version is even slower probably due to the nature of linked list, and the inefficiencies in traversing them. The merge sort in both implementations has a best and worst case of O(nlogn). The data reflects this as it shows the increase in run time is steady but not incredibly extreme towards higher inputs. As it is doubled it increases close to the predicted ratio. The heap sort has the same big-O of the merge sort for its worst and average case but it performs much better towards lower input size than higher ones when compared to merge sort. This is seen clearly by looking at the growing factors of merge and heap sort. The insertion sort in the best case, sorted array, worked phenomenally faster than the worst case, descending order sorted array. The difference was about 500 milliseconds on a 20,000 element array!