AGoldbergLab4

Analysis

The time and space complexities of these algorithms vary depending on the input data and implementation. The big O for all of the quicksort methods is O(n^2) due to the nested while loop that iterates through the stack, but the best case scenarios vary vastly. Quicksort's time complexity is heavily influenced from the choice of pivot, and using strategies like median of three or random pivots can help to improve performance. Merge Sorts for linked lists have a consistent time complexity of O(n*log(n)) and space complexity of O(log(n)). Insertion Sorts have a higher time complexity in the average and worst-case scenarios but a lower space complexity. A hybrid algorithm combining Quicksort and Insertion Sort tries to get the best of both algorithms, with time complexity ranging from O(n) to O(n^2) and space complexity of O(log(n)).

Quicksort's number of exchanges and comparisons depend on the quality of pivot selection. Worst case scenario, when the input data is already sorted or nearly sorted and the first item is chosen as the pivot, Quicksort performs poorly and makes more comparisons and exchanges. On the other hand, Natural Merge Sort consistently performs O(n*log(n)) comparisons. Quicksort's performance is sensitive to the order of the input data. If the input is already sorted or nearly sorted, the worst-case scenario occurs with an a time complexity of O(n^2). In contrast, Natural Merge Sort's performance is not affected by the order of the input data, consistently achieving a time complexity of O(n*log(n)).

As the file size increases, both algorithms take more time to sort the data. However, Quicksort's performance varies significantly depending on the input data's order and pivot selection method, whereas Natural Merge Sort maintains a steady performance with O(n*log(n)) time complexity. Using different pivot selection methods can greatly impact Quicksort's performance. For example, using the median-of-three or random pivot can help avoid the worst-case scenario and improve the average and worst-case time complexity. Additionally, balanced partition sizes lead to better time complexity, closer to O(n*log(n)).

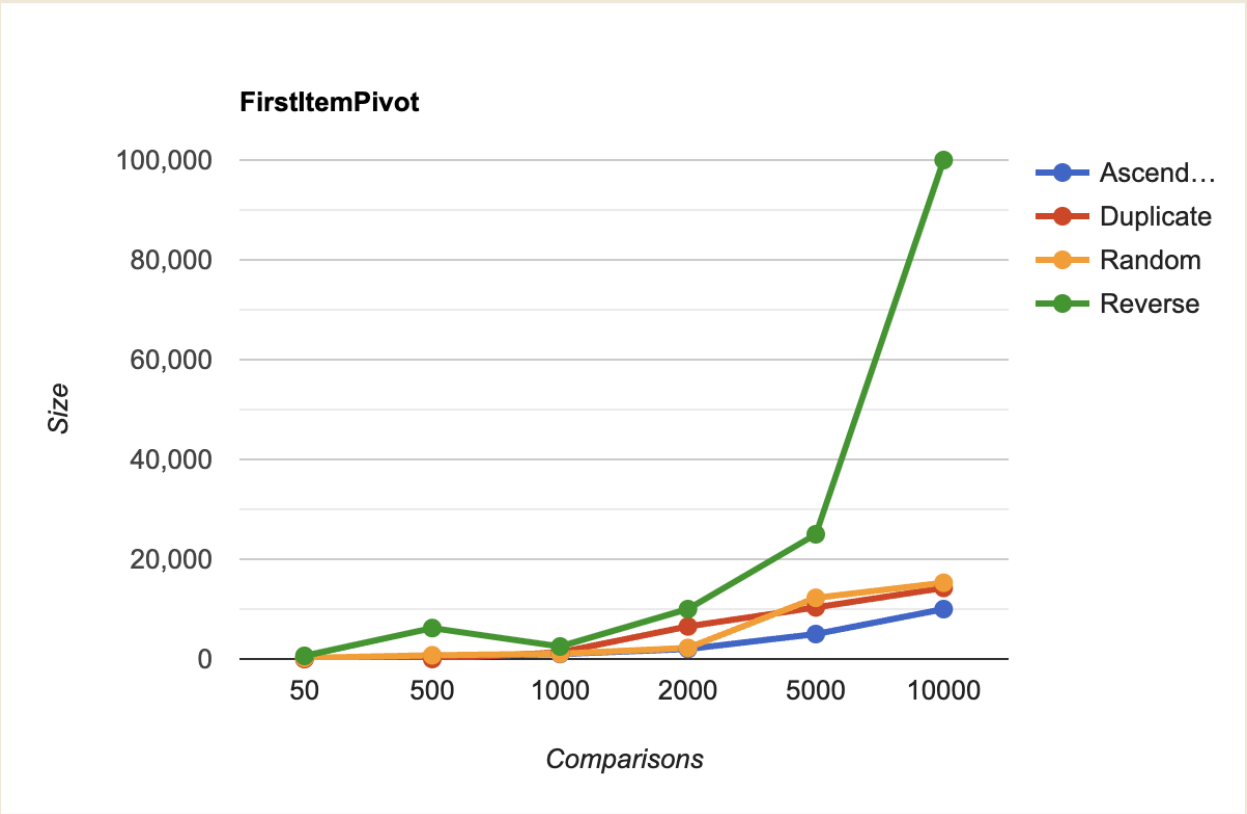
Natural Merge is a sorting algorithm with a consistent time big O of O(n*log(n)) and space complexity of O(log(n)) for linked lists. It performs well no matter the order of input data and has a predictable performance, making it a reliable choice for sorting big datasets. In the table below you can see the comparison between the runs of the median of 3 and mergeSort programs. As we can see, mergeSort has a consistently lower amount of comparisons across the board, albeit a relatively higher number of exchanges. In the graphs below, you can see how to comparisons for each method compare on different data sets. On some of the graphs I removed the worst case scenario so that you can see the difference between the types of data (and the worst case is the same in each graph).

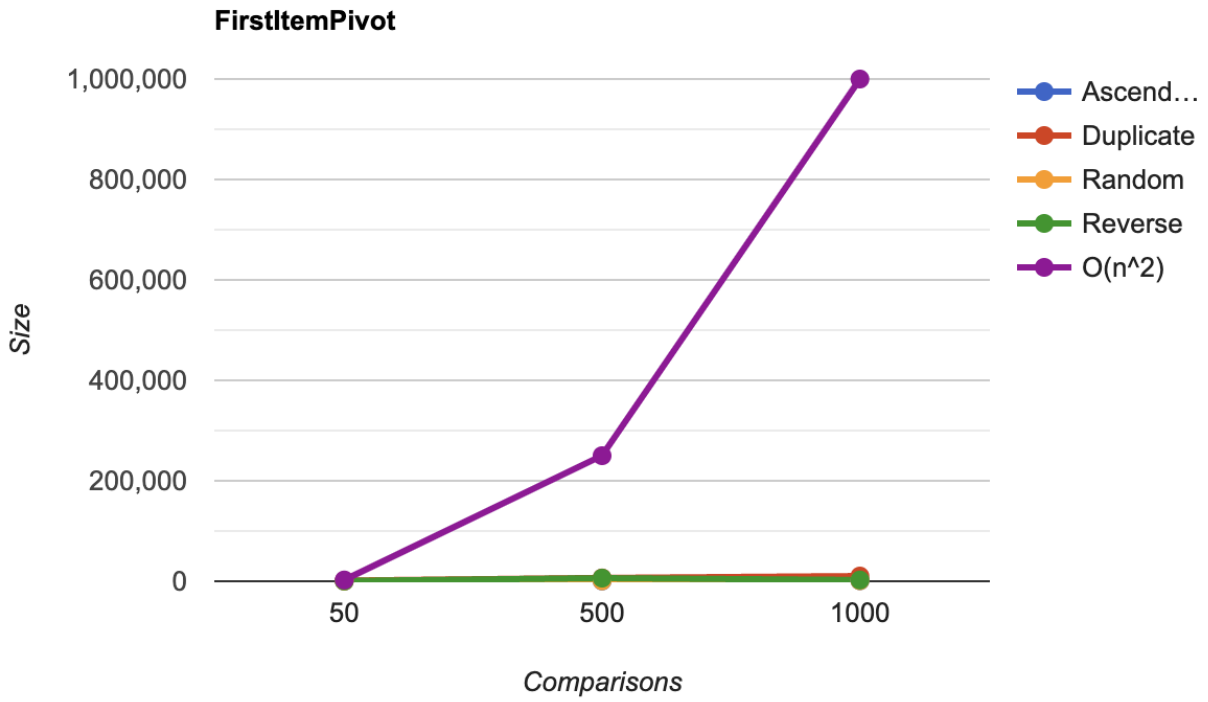
In conclusion, the most crucial factor affecting the efficiency of Quicksort is the pivot selection method, which influences the time complexity and number of exchanges and comparisons. Natural Merge Sort offers consistent performance with a time complexity of O(n*log(n)) and space complexity of O(log(n)) for linked lists. Considering both time and space efficiency, Natural Merge Sort is a good choice, while Quicksort can be optimized with optimized

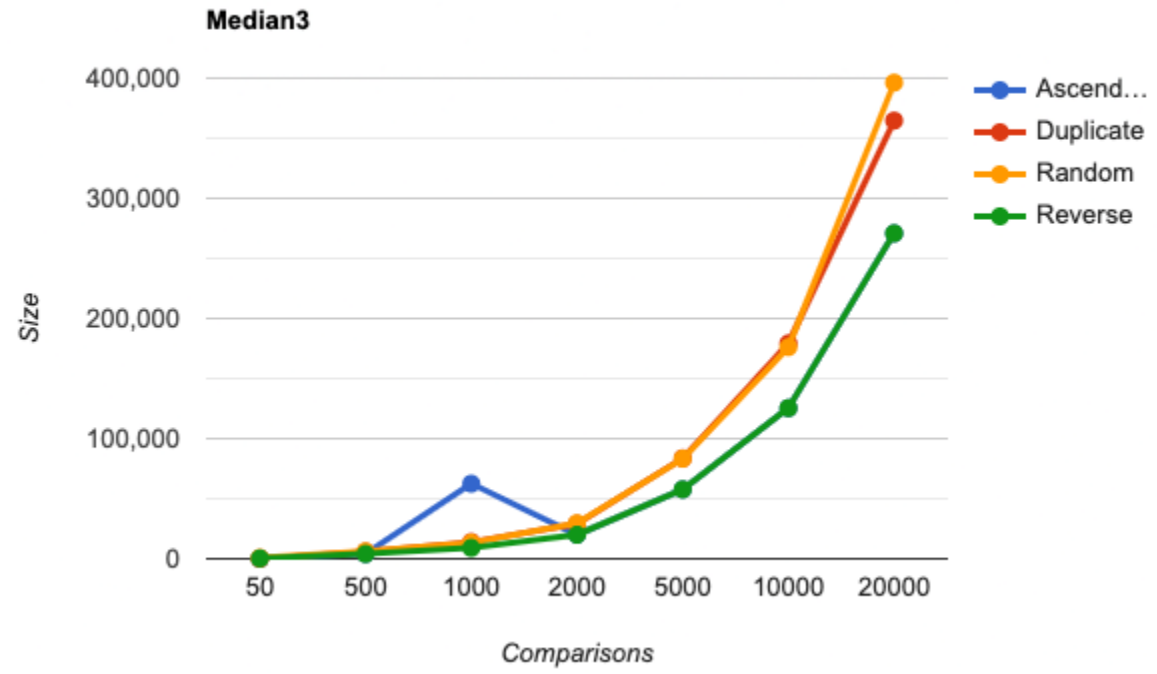
pivot selection. The choice of data structure for Natural Merge Sort allows for better space efficiency compared to array based implementations.

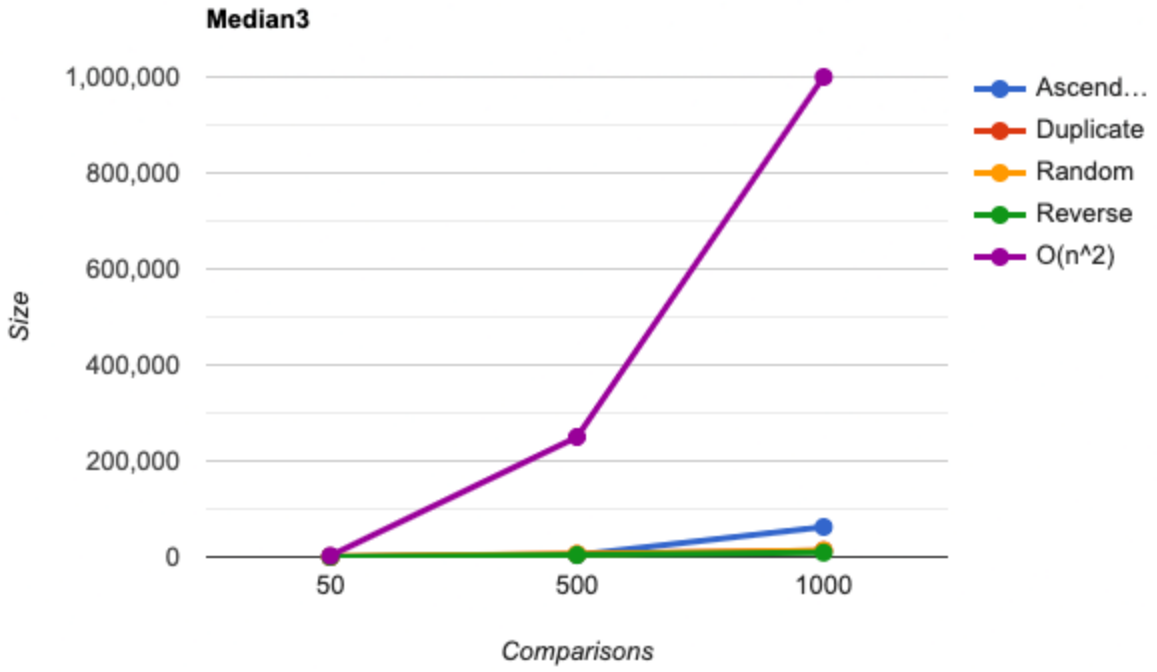
| ascending | Median of 3 | MergeSort |
|---|---|---|
| 50 | Comparisons: 255, Exchanges: 31 | Comparisons: 153, Exchanges: 153 |
| 500 | Comparisons: 4008, Exchanges: 255 | Comparisons: 2272, Exchanges: 2272 |
| 1000 | Comparisons: 62500, Exchanges: 62500 | Comparisons: 5044, Exchanges: 5044 |
| 2000 | Comparisons: 20010, Exchanges: 1023 | Comparisons: 11088, Exchanges: 11088 |
| 5000 | Comparisons: 57726, Exchanges: 2952 | Comparisons: 11088, Exchanges: 11088 |
| 10000 | Comparisons: 125439, Exchanges: 5904 | Comparisons: 69008, Exchanges: 69008 |
| 20000 | Comparisons: 270864, Exchanges: 11808 | Comparisons: 148016, Exchanges: 148016 |
| duplicate | | |
| 50 | | |
| 500 | Comparisons: 5766, Exchanges: 1189 | Comparisons: 3842, Exchanges: 3842 |
| 1k | Comparisons: 13967, Exchanges: 2593 | Comparisons: 8670, Exchanges: 8670 |
| 2k | Comparisons: 28995, Exchanges: 5747 | Comparisons: 19417, Exchanges: 19417 |
| 5k | Comparisons: 83651, Exchanges: 15763 | Comparisons: 55324, Exchanges: 55324 |
| 10k | Comparisons: 179316, Exchanges: 33739 | Comparisons: 120413, Exchanges: 120413 |
| 20k | Comparisons: 364669, Exchanges: 73096 | Comparisons: 260891, Exchanges: 260891 |
| random | | |
| 50 | Comparisons: 768, Exchanges: 190 | Comparisons: 477, Exchanges: 477 |
| 500 | Comparisons: 6218, Exchanges: 1179 | Comparisons: 3854, Exchanges: 3854 |
| 1k | Comparisons: 13335, Exchanges: 2564 | Comparisons: 8705, Exchanges: 8705 |
| 2k | Comparisons: 29613, Exchanges: 5670 | Comparisons: 19436, Exchanges: 19436 |
| 5k | Comparisons: 83329, Exchanges: 15776 | Comparisons: 55160, Exchanges: 55160 |
| 10k | Comparisons: 176128, Exchanges: 33590 | Comparisons: 120483, Exchanges: 120483 |
| 20k | Comparisons: 396271, Exchanges: 71414 | Comparisons: 260994, Exchanges: 260994 |
| reverse | | |
| 50 | Comparisons: 258, Exchanges: 55 | Comparisons: 133, Exchanges: 133 |
| 500 | Comparisons: 4014, Exchanges: 504 | Comparisons: 2216, Exchanges: 2216 |
| 1k | Comparisons: 9016, Exchanges: 1010 | Comparisons: 4932, Exchanges: 4932 |

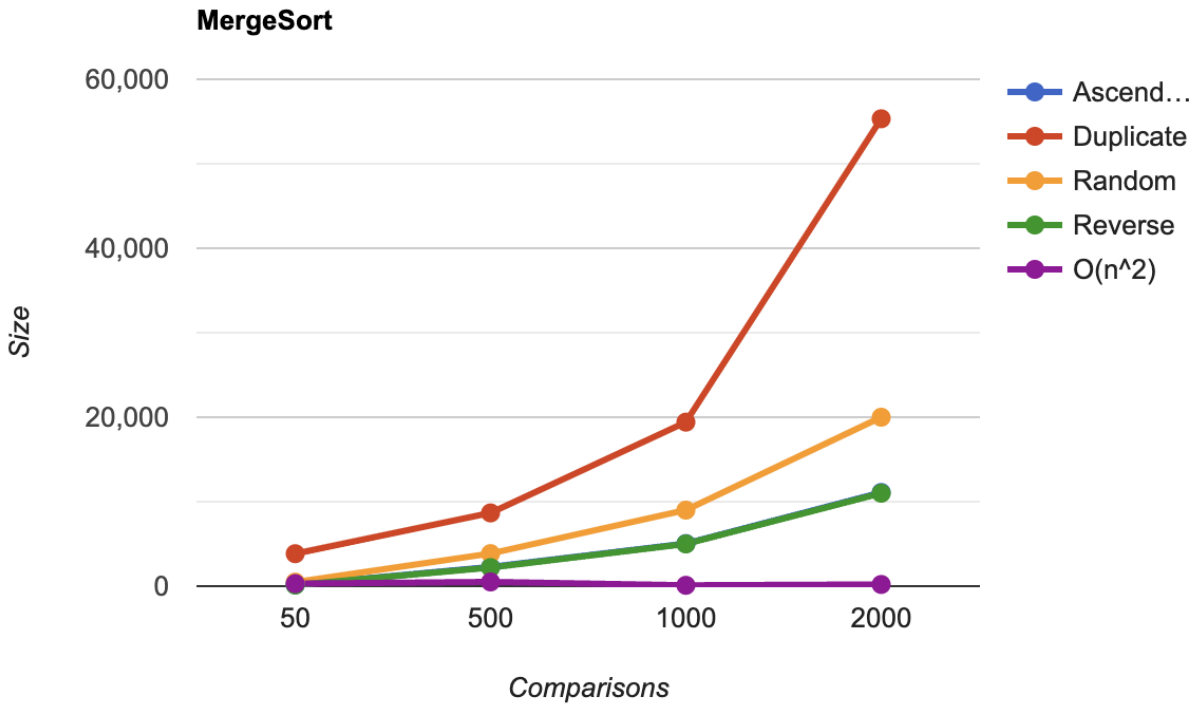
| | | |
|---|---|---|
| 2k | Comparisons: 20018, Exchanges: 2022 | Comparisons: 10864, Exchanges: 10864 |
| 5k | Comparisons: 57738, Exchanges: 5452 | Comparisons: 29804, Exchanges: 29804 |
| 10k | Comparisons: 125452, Exchanges: 10904 | Comparisons: 64608, Exchanges: 64608 |
| 20k | Comparisons: 270878, Exchanges: 21808 | Comparisons: 139216, Exchanges: 139216 |

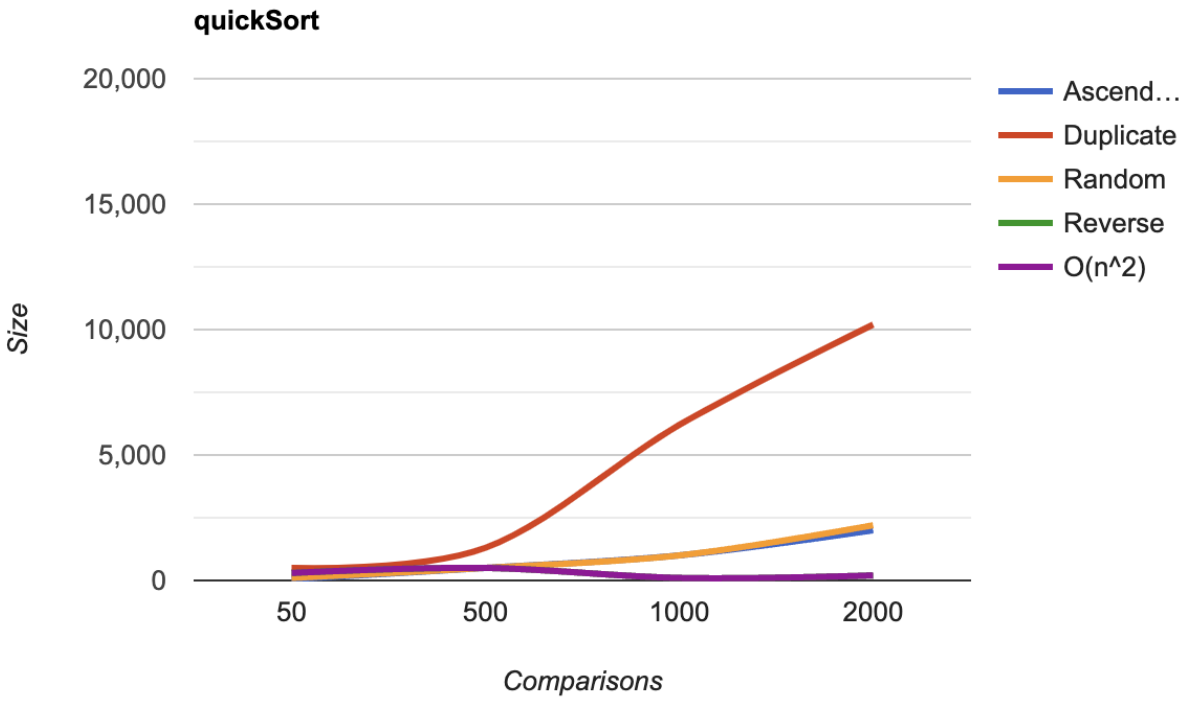



FirstItemPivot
1,000,000
800,000
600,000
400,000
200,000
0
Size
50
500
1000
Comparisons
Ascend…
Duplicate
Random
Reverse
O(n^2)


Median3
400,000
300,000
200,000
100,000
0
Size
50
500
1000
2000
5000
10000
20000
Comparisons
Ascend…
Duplicate
Random
Reverse


Median3
1,000,000
800,000
600,000
400,000
200,000
0
Size
50
500
1000
Comparisons
Ascend…
Duplicate
Random
Reverse
O(n^2)


MergeSort
60,000
40,000
20,000
0
Size
50
500
1000
2000
Comparisons
Ascend…
Duplicate
Random
Reverse
O(n^2)


quickSort
20,000
15,000
10,000
5,000
0
Size
50
500
1000
2000
Comparisons
Ascend…
Duplicate
Random
Reverse
O(n^2)