Drew Smith

CS 124

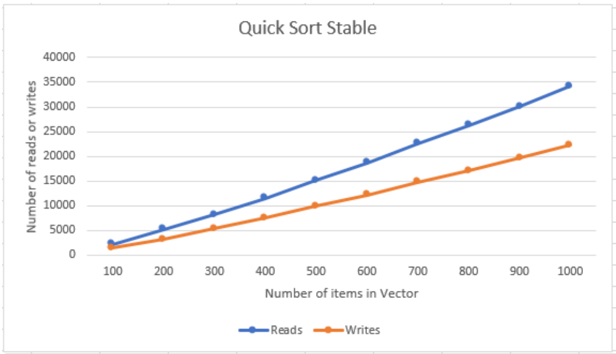
Project 4

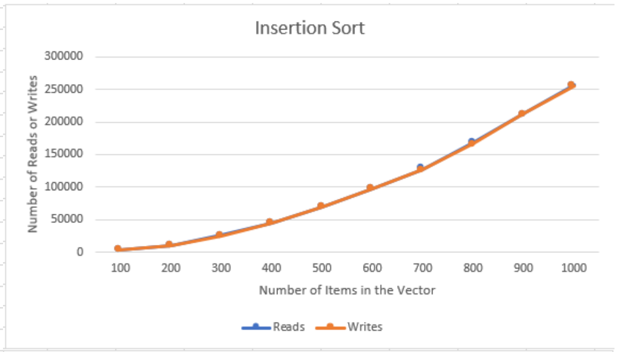
Nov 11, 2020

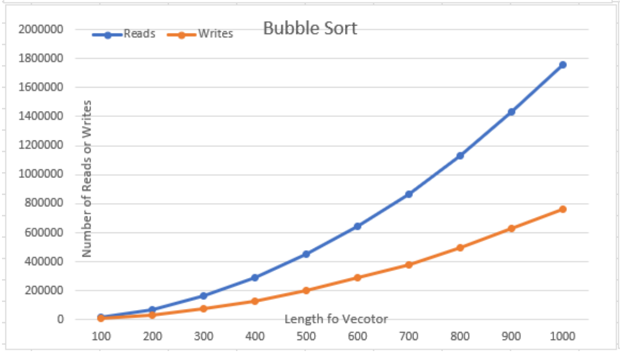
My data set is the results of all the professional tennis tournaments that happened in the year 2000. The six item from this data set that I am representing in my class are the name of the tournament, the name of the match winner, the level rating of the match, the age of the winner, the score of the sets played, and my unique Identifier is created using a combo of the date the matches happened as well as the match number. This made comparing my data slightly more difficult I ended up using some code I found on stack overflow to help me break this data apart into the most important parts. Simply the date and the match number this is due to the year always being 2000. A game is considered greater than another if its match number is higher or if it is later into the year.

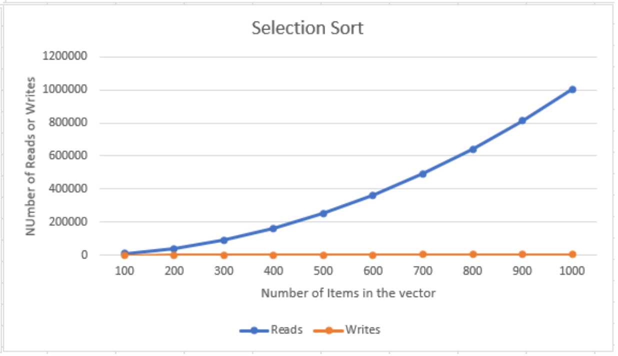
- Tournoment (string type, name of the tournament the matches are in)
- Winner (string type, the name of the winner of the particular match)
- Level (string type, the level rating of the tournament the matches are in)
- Age (float type, the age of the players is saved as a float)
- Score (string type, the score from as many as 5 matches to determine the winner)
- uniquiIdentifer (string type, combo of the date and match number)

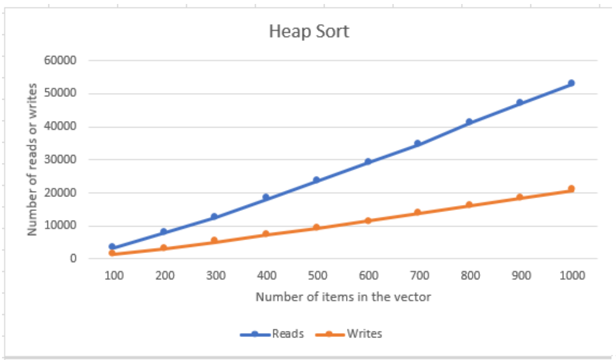
I chose this data set because I have always been interested in sports and tennis was a sport I new very little about so I saw this as not just an opportunity to learn more about computer science but to expand my understanding of a sport I knew very little about.

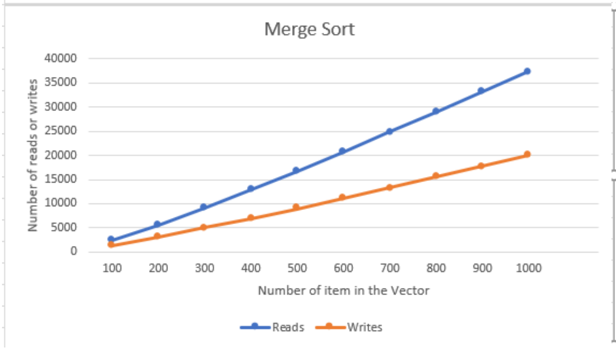


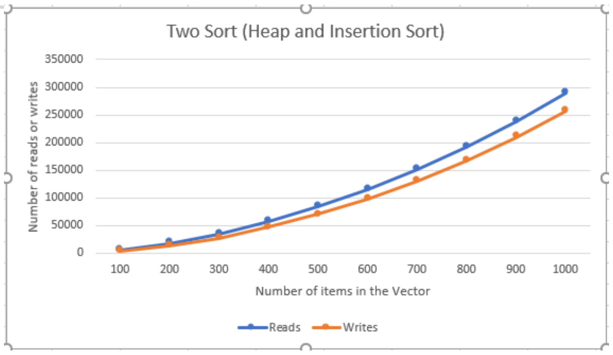












Complexities:	Time	Space
Bubble Sort	O(N^2)	O(1)
Selection Sort	O(N log(N))	O(1)
Insertion Sort	O(N^2)	O(N)
Merge Sort	O(N)	O(N)
Quick Sort Stable	O(N log(N))	O(N)
Heap Sort	O(N)	O(1)
Two Sort	O(N)	O(N)

The algorithm that would be most efficient algorithm goes it would be Merge sort, while technically quick sort can be better in a worst-case scenario the complexity of quicksort becomes much worse the that of merge sort. This along with the number of reads and writes being relatively low compared to something such as bubble sort or insertion sort is why I would place merge sort as the most efficient algorithm.

If you need to sort a contacts list on a mobile app, which sorting algorithm(s) would you use and why?

Through a few minor assumptions the conclusion I have come to in this question is the best algorithm to use would be merge sort. Simple because it is very effective along with being very efficient it has low reads and writes comparatively. This is however assuming that space is not problem and the algorithm must be stable. However, if the algorithm does not need to be stable the best option would become heap sort for its constant space complexity. If space I required to be as low as possible while still being stable the only option would have to bubble sort while it is incredibly inefficient it does have constant space complexity while still being stable.

What about if you need to sort a database of 20 million client files that are stored in a datacenter in the cloud?

For this question the number of times the cloud storage datacenter would be the most crucial. The best algorithm to keep the number of reads as low as possible would have to be merge sort this is because along with it being efficient it also has the lowest number of reads among all the algorithms that I tested in this assignment. This is all operating under the assumption that space would not be a problem and the client would be willing to sacrifice on space complexity. I would say this is a fair assumption because there are already 20 million client files being stored.

Extra Credit:

For this assignment I used seven algorithms instead of the minimum requirement of 5.