Tyler Scott

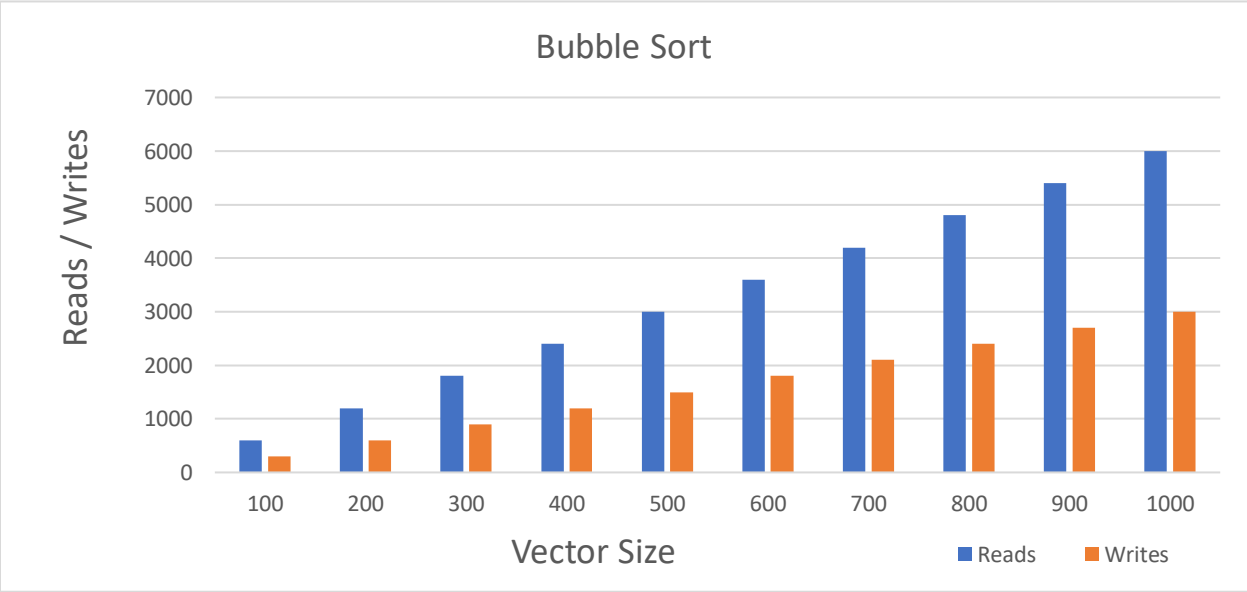
CS124 Project 4

Professor Dion

The data set I have been using for my projects is the weather is the weather on Mars collected by the curiosity rover during a five-year period. For this project I am organizing the set by the *pressure* field, which is the atmospheric pressure of that specific time. I chose the pressure field because, though it stays roughly in between 700 – 900, the numbers are in no particular order themselves. For the two-sort algorithm I used the max temperature field as well. They way I structured the code was to separate each algorithm out into its own class and then create a header file where all the operations with the algorithms are performed. They are then initialized and called upon in main.cpp. To gather the total number of reads and writes in each algorithm I created a small function that just takes in two global variables and sums them with the current total of each. This function is then called each time there is either a comparison or an assignment on a Comparison generic type. There are also getters to retrieve the read and write numbers to be output to a file.

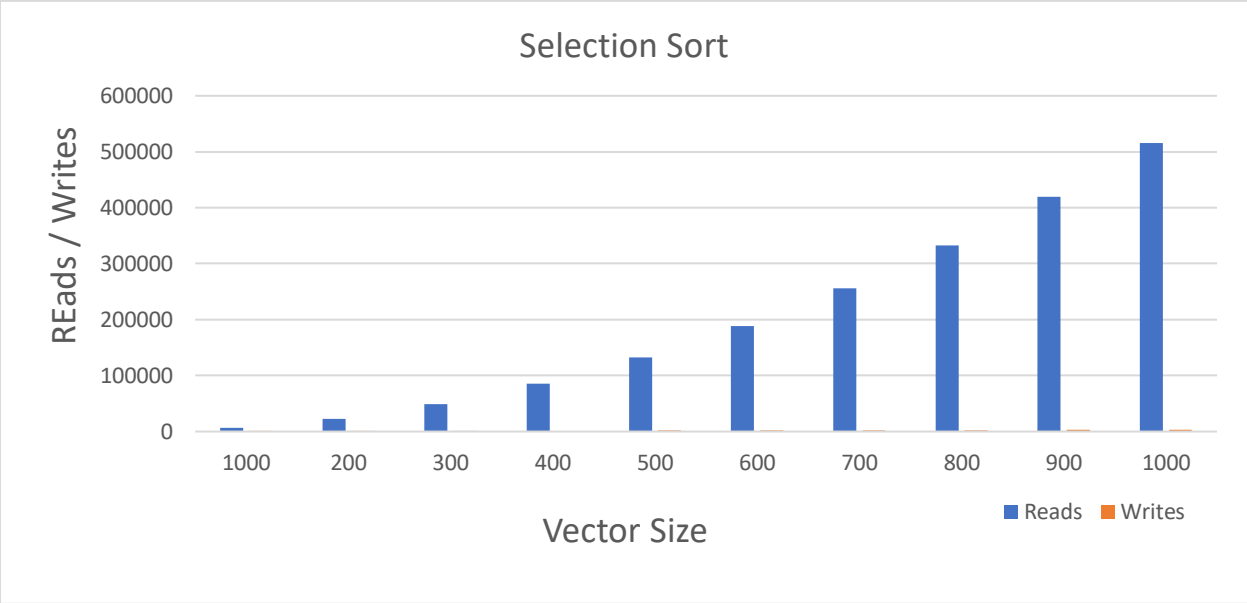
Bubble Sort

Bubble sort was first and also one of the more straightforward algorithms. The algorithm appears to grow by a direct multiple of the number of comparisons. For example I counted 6 reads in total in the algorithm and then sorting 100 gave 600 read in total, sorting 200 gave 1200, 300 gave 1800, and so on. Bubble sort has the complexity of O(N^2) as evidenced by the nested loop and if statement in the algorithms code. This first table shows the growth of reads and write of bubble sort:



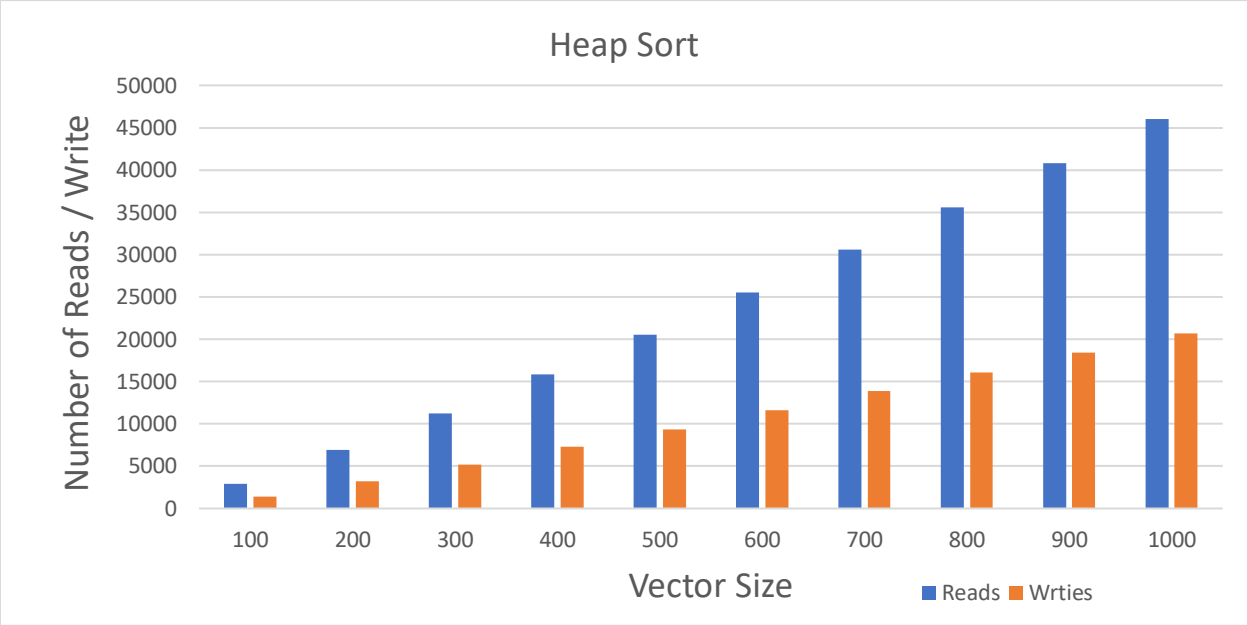
Selection Sort

Selection sort was definitely through me for a loop at first as the number of reads and writes I was getting was substantially larger than that of bubble sort. I think the justification for this would be that selection sort simply iterates through the entire vector many more times than bubble sort does before it makes its swaps. Through research it appears that an equation to estimate comparisons of selection sort is N(N-1)/2, which after substituting N with 1000 does actually yield 500,000. This leads me to believe some of my assumptions are accurate. As the next chart shows the number of reads is so much larger than the number of writes they are almost negligible. This indicates that the sorting is comparing way more elements than it is swapping anything. This may be circumstantially more efficient than bubble sort as possibly with a small enough vector the speed boost of having to write less information may become beneficial. Though selection sort also has a complexity of O(n^2) so it's hard to say it is explicitly more efficient than bubble sort.



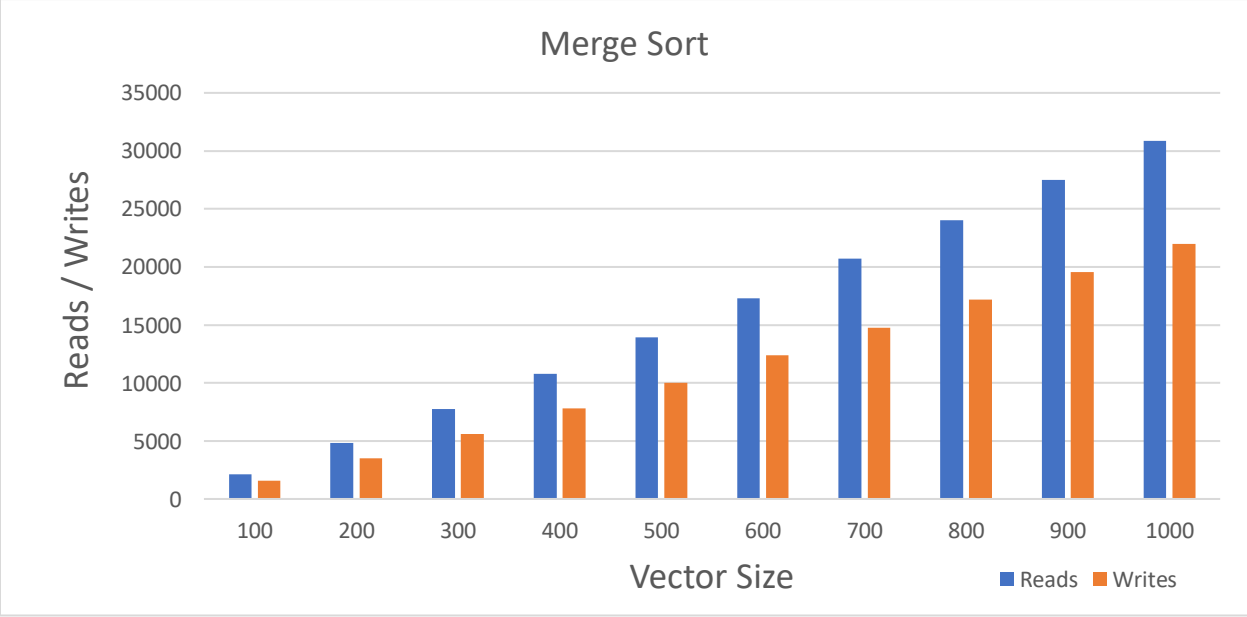
Heap Sort

Heap sort right off the bat seems to be a fairly efficient algorithm. It appears that the growth could possibly be exponential. The number of reads and writes I am getting increases by 3000, then 4000, then 5000, etc, which may not be the best for large sets of data. I would expect that pattern would continue as it grows larger. It also looks, basing off of the chart, that the difference between the number of reads and writes will increase as the size of the vector grows. I feel that indicates heapsort would be a good algorithm for searching larger arrays and vectors, at least more so than bubble and selection sort.



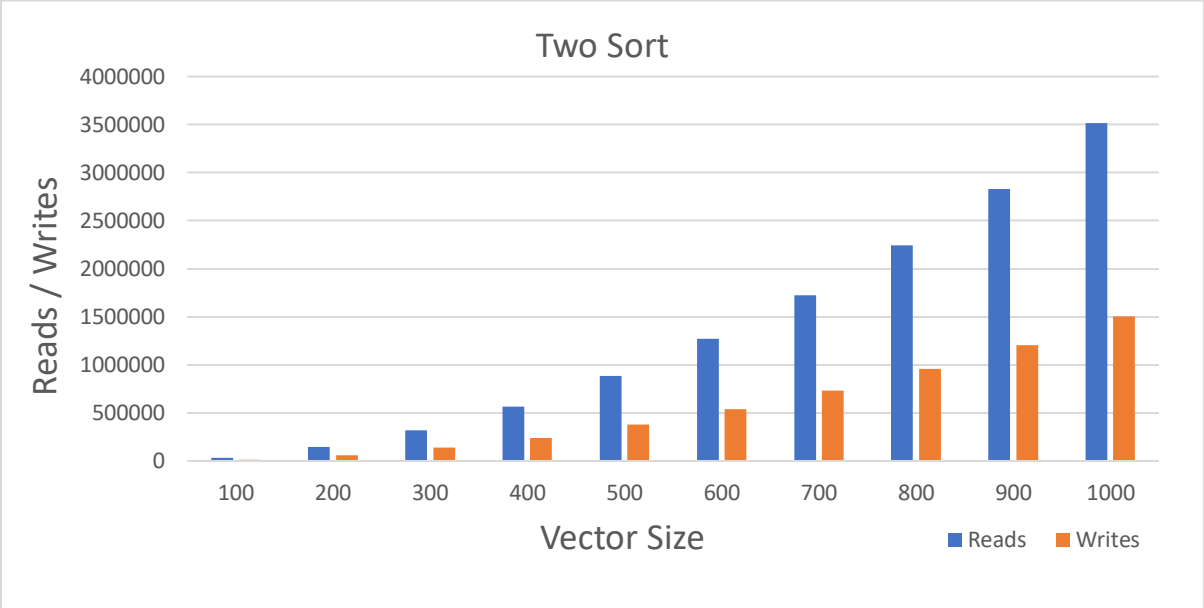
Merge Sort

Merge sort also seems to be a decently efficient algorithm in comparison to bubble and selection sort. One problem I had read at (http://www-cs-students.stanford.edu/~ rashmi/ projects/Sorting) is that merge sort uses more memory than other sorting algorithms. Looking through the code I would say this is a product of using recursive methods with temporary vectors. Having multiple vectors being created and searched would certainly increase the amount of memory needed to run the application. I feel that this explains why the graph shows more writes for merge sort when comparing to heap sort. I believe that both merge sort and heap sort's complexity would be O(N log N).



Two Sort

The second field I sorted by for two sort is the maximum temperature in a day on Mars. I decided to use selection sort for the unstable algorithm while using bubble sort for the stable one. I slightly modified bubble sort to take in my own MarsWeather data type and make comparisons on the max_temp field. The number of reads and writes seem to have kept along a similar trend that selection sort did and yielded pretty high results. My first thought was that this is likely not the best way to sort, though I can see it as almost a balance between bubble and selection sort. It seems to have kept a larger difference between reads and writes than bubble sort, but may use up more memory than selection sort would. I'd be inclined to believe this would keep O(N^2) complexity similarly to the others.



Conclusions

For a large set of data, for example sorting a database of 20 million clients, I think I would likely choose merge sort. Merge sort seems to have a good balance between the actual number of comparisons and the number of assignments as well as having a lower number overall than some of the others. Though it does use more memory than heapsort I think the slight boost in speed would be worth it with such a large set of data. For a small set of data like a contacts list on a phone I would probably go for selection sort. My thinking is with its low number of writes it would be quickest and wouldn't be a waste of resources, like memory, as it would be with merge sort. Alternatively, heap sort could be a good medium between speed and memory that would apply to either scenario.