Alberto Chi Kan Ng
CPSC-350-02
Professor Rene

Sorting Algorithms Report:

The time differences are not as drastic as I expected. I do believe that this is due to the fact I did not test the 5 sorting algorithms with a larger file. The file I used only consisted of 8 doubles, one on each line. However, I was able to see possible patterns. One pattern was quick sort tends to spend the most time on sorting. Each sorting algorithm has its strength and its weaknesses. For example, the trade off for an easily implementable selection sort would be the runtime of O(n^2). Another example would be bubble sort, bubble sort is extremely efficient if a list is partially sorted, however, if it is not partially sorted, the worst possible runtime will be O(n^2) as well. The last example would be merge sort. Unlike a lot of sorting algorithms, the worst possible runtime for merge sort is O(nlog(n)). So if someone were to look for a stable range of runtime, merge sort would be beneficial. As for the trade off would be additional memories are needed.

The choice of programming languages can also affect the results. For example, languages like java and python do not require garbage collection, this may cause the efficiency to go down. Also, languages like java do not compile directly to machine code, this may also cause the efficiency to go down. The efficiency of different compilers may also contribute to different runtime. One shortcoming I can think of is that in the real world runtime and space complexity are not the only two deciding factors. Some other factors people might be concerned with are how easy it is to maintain/implement the algorithm, and if a complex algorithm is needed for the scenario. An example would be a backup database. Usually, as long as it is not too slow and it works, it serves the purpose of maintaining the backup database. Also, it would be counterproductive if someone were to implement sorting algorithms AI uses into a backup database.