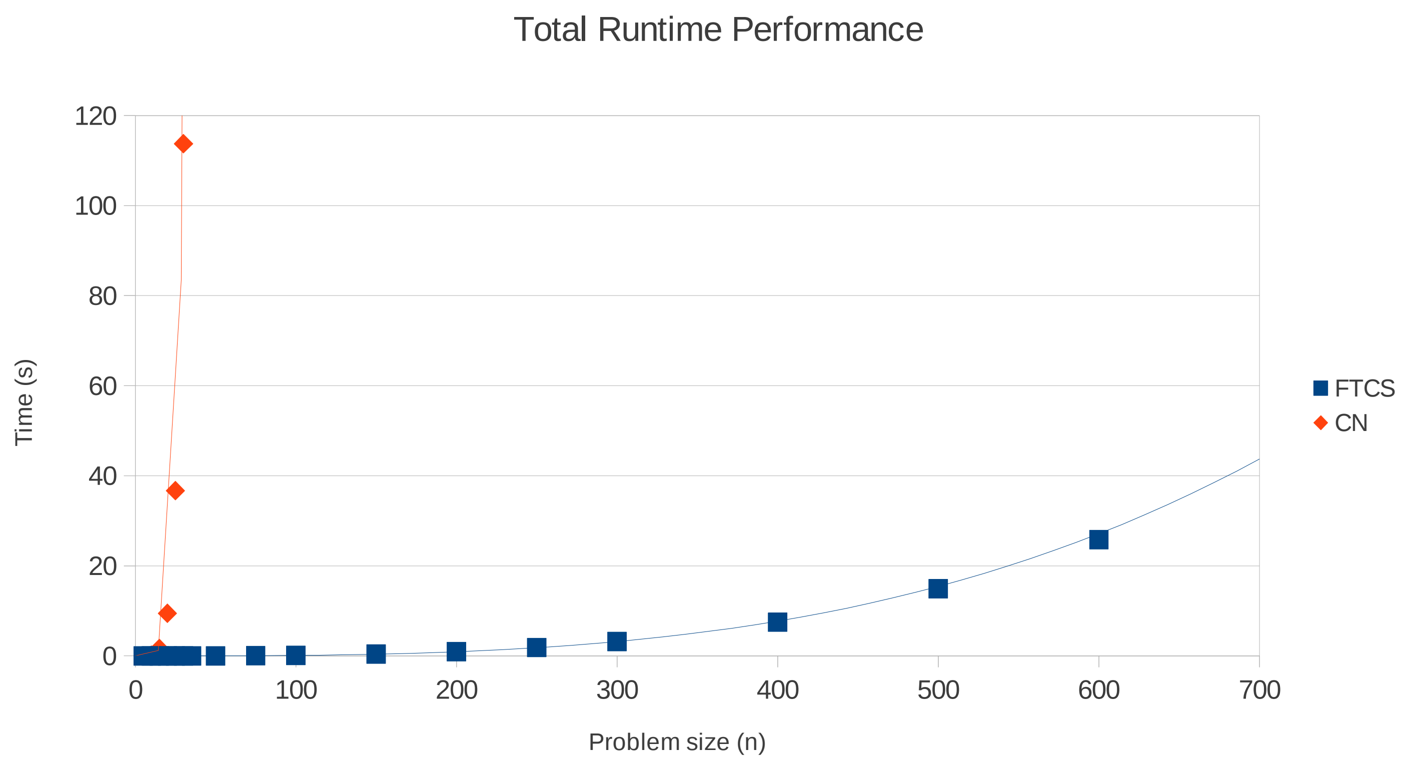

The above graph shows runtime per algorithmic timestep for a problem size in terms of n, where n is the dimension of a cubic matrix (sized nxnxn). It is based off the below data I gathered from running my code at various problem sizes. All problems were run with the default settings of my program, only changing the problem size.

	Time (s)	
Problem size	FTCS	CN
5	0	0
10	0	0.14
15	0	1.63
20	0	9.46
25	0	36.69
30	0	113.75
35	0	-
50	0.01	-
75	0.05	-
100	0.11	-
150	0.4	-
200	0.94	-
250	1.86	-
300	3.16	-
400	7.48	-
500	14.92	-
600	25.81	-

Note that I ran out of memory for FTCS when above n=600, and for CN above n=30.

Conclusions

It's blatantly clear that my FTCS implementation is several orders of magnitude faster than my CN implementation, and consumes much less memory per problem size. My CN implementation could be made substantially better by accounting for the sparsity of A, and representing it in terms of 7 vectors, rather than a full matrix. This would allow me to avoid using full gaussian elimination, which would make it run faster and use a lot less memory.

However, since this requirement was waived, I didn't spend the time implementing it. Furthermore, it would still be much slower than FTCS. The only upside to using CN is that is unconditionally stable, as opposed to FTCS which is not.