Part 4

1. Decreasing the initial estimate t0 led to the program doing many more context switches, especially early on before the estimates seemed to really reflect burst times. Oppositely, setting the initial estimate t0 high made the program much more likely to not context switch as often as early. It was interesting to set alpha to 0 and notice that the estimate would always be equal to t0. Or if you set alpha to 1, you could see that the estimate reflected whatever the previous burst was. Setting an alpha value of 0.5 had similar behavior to setting either the t0 or the alpha low and the other setting relatively high.
2. Our approximation is only as good as our data. If, for example, the process had bursts that were very different from each other, our implementation would do a better job of preventing starvation, but a worse job of supporting a proportional amount of run time for different processes. For somewhat small data sets with burst sizes that don’t differ too much from one to the other, I would say our implementation does a fine job representing those scenarios. Knowing the precise run time would optimize the ratio between starvation management and proportional CPU time, but without knowing the next burst time precisely, our version is able to do an alright job splitting up the jobs. Our implementation prevents starvation (usually) and does alright to divvy up run time, so I think it is a pretty good approximation.

Part 5

1. A
2. B
3. The most obvious reason I see for not using a lottery is that processes will avoid starvation, and in some cases, the process should probably just starve compared to others. To use our example from class, if your OS constantly has high priority processes trying to run, then a process which calculates the digits of pi should not get CPU time until it is the only thing running. When there are much more important things to run, a lottery style system disrupts their response and turnaround times so that it can prevent any one process from starving.