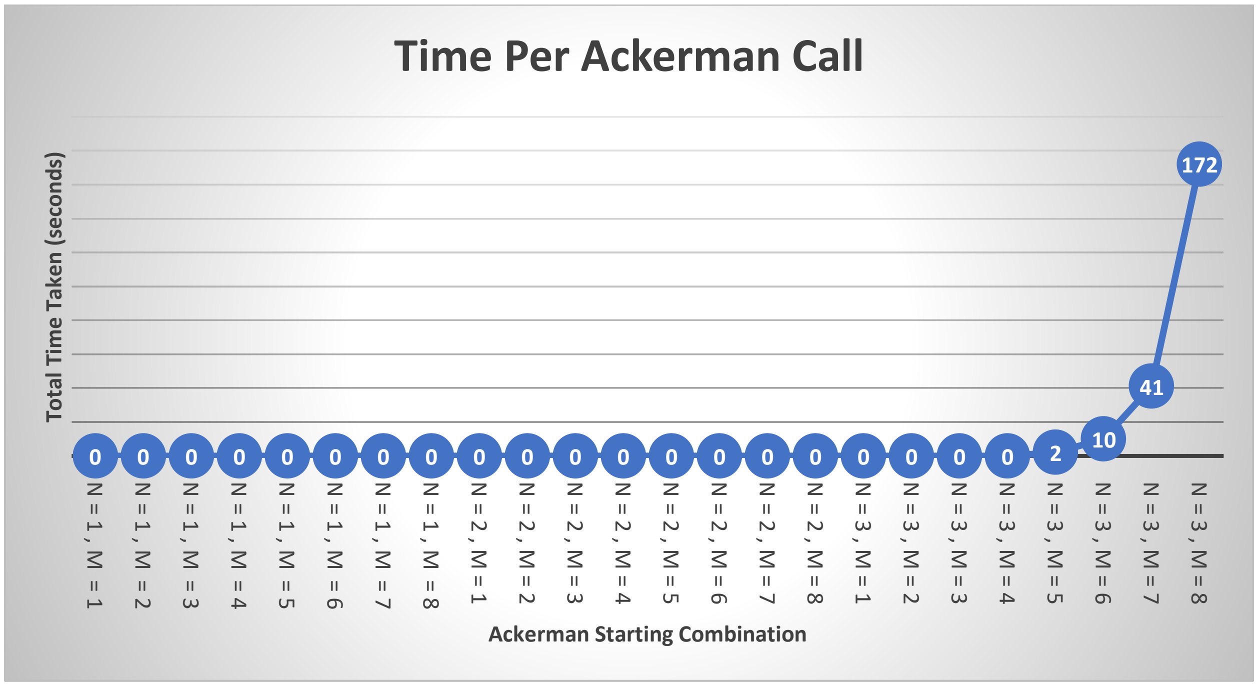
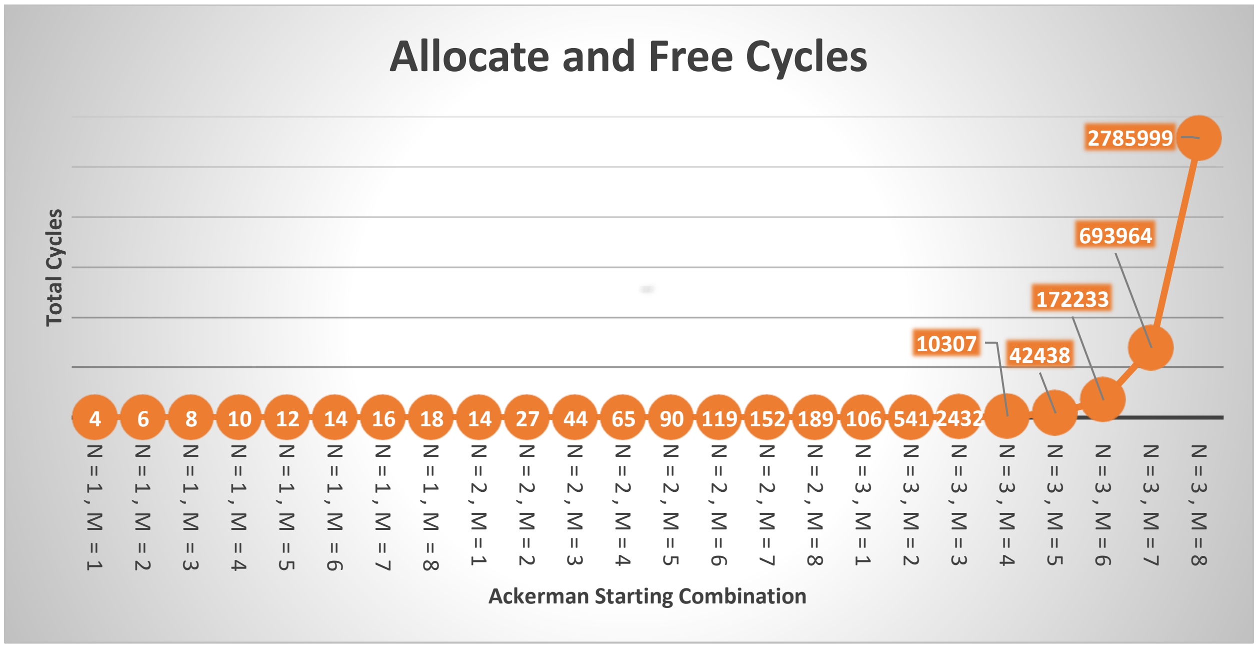
Sean Neal
9/20/2019
Section 512

PA1: Buddy System Memory Allocation



I was surprised to see the program report back taking zero seconds for so many of the test cases. However, the more notable aspect of the graph is the exponential growth in time. This is similarly reflected in the amount of allocation and free cycles performed by the program.



It is apparent that the sharp increase in time taken to run the program comes from the exponential growth of the allocate and free cycles starting once n = 2. All data points with n = 1 grow in cycles and thus time linearly.

Sean Neal
9/20/2019
Section 512

I believe the program is bottlenecking when allocating the appropriate memory and splitting the blocks individually. Depending on how frequently memory is being allocated and freed, it can easily had many processes to keep splitting the memory into buddies.

One modification I would make to the buddy system allocation to improve performance would be to begin the system in a state already half split. In the lifespan of a program, it is not consistently needing the most nor least amount of memory accessible consistently. Thus, by starting with blocks available for mid-sized data, time would be saved from having to split all the way down to the basic block size, merge all the way up, only to split down to the basic block size again.

I believe my program's efficiency could be improved by altering how my alloc function handles the freelist. Currently, I am removing one element out of the freelist, splitting it and inserting the two split blocks into the freelist indexes below. However, I believe it would be possible to run through each index, decide the splitting and removing actions without preforming them, then modifying the freelist at the end of the alloc function once we know what it should look like. This would theoretically save many calls to remove, split and insert, saving all of the calls for the end rather than each iteration. This is of course because the less lines of code being processed by the computer, the quicker it can finish the program's process. Each function being called in the loop of my alloc function contains many lines of code, all being run over and over. Calling the functions at the end only once to set up the freelist would save running many lines of code. Thus, removing these function calls would greatly improve the efficiency of the alloc function and the program.