Kery Zhang-Section 503
Cesar Ortega-Section 501

Analysis of my_allocator.c

The purpose of this project was to create a memory allocator based on the buddy system, in which you split a chunk of memory into half until you get the smallest possible chunk able to fit whatever you're trying to store and then whenever it gets freed, it will merge the unused memory back into a full chunk. As, we increased the m and n variables in Ackermann, the time it took to run would increase by a large factor, definitely not linearly. We did not time it exactly, but we speculate that it is exponential or even above exponential due to the recursive nature of the function.

We were not able to get our code completely working. We believe there is an issue in either my_malloc() or my_free() which causes a segmentation fault after many allocation and free calls in ackerman(3,6). With some internal counting, we figured this number to be 36 allocations and 21 frees. In the 31st free it seg-faults due to an error in the address it is given. For simple tests of allocation and freeing, the functions do their job splendidly, however for rapid allocation and deallocation, moreover in random locations throughout memory, it ran into issues. We believe that everything else is working besides that and thus we were not able to completely test the Ackermann function.

Had we gotten it working, we speculate that the biggest bottleneck would be trying to insert/free various small blocks. If you try to insert or free small blocks, the function would have to call the merge/split function multiple times to allocate/merge that block. Additionally, fragmentation can become a problem if there are multiple blocks with different sizes so that the memory is not contiguous, which could affect locality. To improve the performance, we could only free at certain times when the memory is almost full or completely so that it lowers the amortized amount of merges. Our own computer actually does this on our hard drives when we aren't actively using it through the disk defragger, although that's with a hard drive and we're using memory. The initial splitting function also takes a long time due to its recursive nature. If the initial split call requires a node of the smallest allowable size, it will first move all the way up to the first node (the one that is currently the size of the entire memory block) and then start splitting from there.

In conclusion, while we were not able to finish, we were able to understand the problem and its challenges. The Ackermann function grows very quickly, even more quickly than exponential. However since we cannot completely test the function in its entirety because we were not able to finish the merge function. The merge and split function can take a lot of time and thus reducing the amount of times we merge and split could help our time performance, but it could potentially hurt locality due to the function "defragmenting" less often.