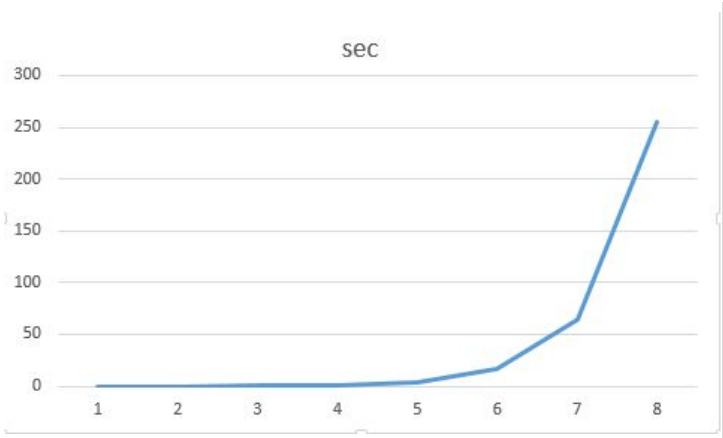
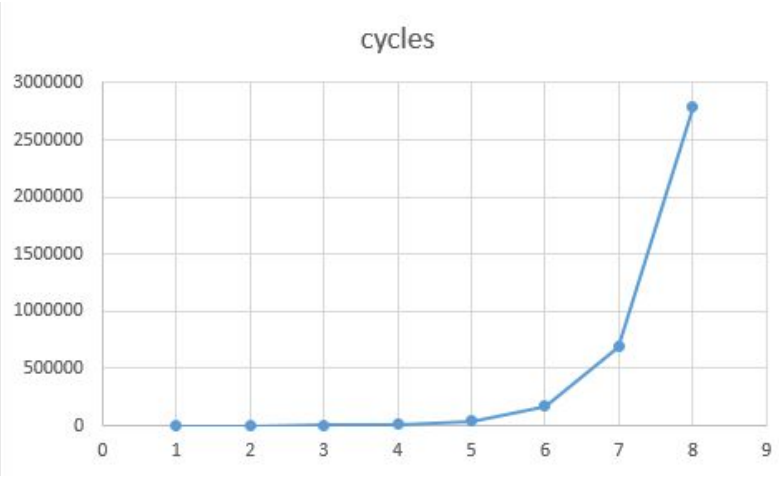
Machine Problem 2 Report

Authors: Clifton Sims, Chris Comeaux

One major bottleneck in our implementation is in our merge function. It not only must find buddy pairs, but it must also deal with the order the free elements. Since it is called recursively, at the worst case, it must go through the entire free list and every element linked to the free list to merge the block correctly. When the block size is large, this operation is very expensive. One way we can improve the functionality is to wait until we run out of memory to call merge. This prevents the merge operation from running every time we encounter 2 free blocks but instead only runs when we need more memory. Although it might make the merge operation more complex, it will decrease the number of times we call merge which, in turn, will help speed up performance.

As we see from the graphs below, the higher the m value, the longer it takes the algorithm to finish. In other words, the more allocate/free cycles the Ackerman function performs the longer the it takes for the algorithm to terminate with our implementation of my_malloc.





For both of the Graphs above, we assume n = 3 while m is constantly incrementing by one for the Ackermann function. From the data collected, we can see from the lines displayed that there is a correlation between how long it takes for the program to run and how many cycles it goes through when it's finished.