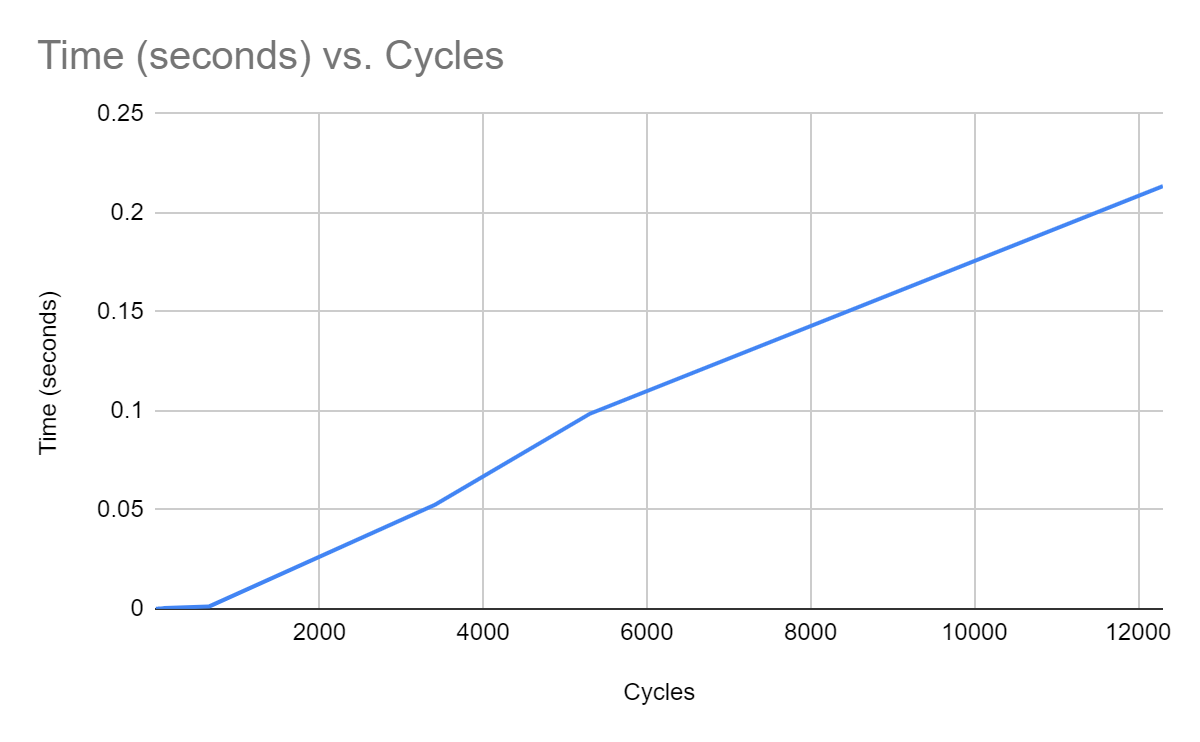
Val Robichaux

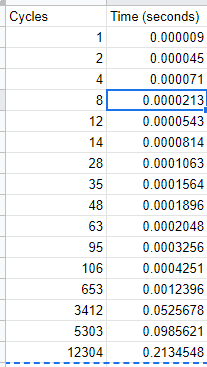
CSCE 313 - 503

Dr. Tanzir Ahmed

2 February 2020

Buddy System Memory Allocator

The buddyAlloc system is a universal system used today to teach the understanding of memory paring and allocation. I implemented my buddy allocation system in the hopes or creating a simple, fast, and effective method. I think that there are still many areas that I could improve in, in order to create a faster approach, and get quicker time values for all ackermann values. Pictured below is the graph of Cycles vs. The time is took to complete the ackermann values. From first glance it is clear to see that I have a very linear relationship with the points that are listed on the graph, with only a few errors coming from the mid-range values of my ackermann function values.

Seen here are all the ackermann times and values that make up the graph that is shown above. With my current implementation, I can see that in the middle range values of the ackermann function such as (2,5) (2,7) and (3,1) my program starts to bottleneck due to the splitting and merging the blocks. Since by default there is a vector that stores the linked lists, any time that there needed to be a merge or a split, my code would have to traverse the entire vector and linked list in order to go find that block and produce the expected output. At middle values like these, this starts to be a hard thing to do since the linked list is already populated with so many values. Finding the right blockheaders takes a very long time in this case. A possible solution for this would be for me to have made a hash table in order to store all of my linked list, that way there is a direct method of obtaining blockheaders and there is no need to traverse any linked list. It would be immediate retrieval, resulting in a run time of O(1) instead of O(log(n)). With the functions that were implemented in the assignment, all of them have performance issues due to their runtime being O(n) and not a constant. Every function must refer back to the free list whether it is Merge, Split, Alloc or Free. Every single one of these functions must point back to the freelist to find the given block header that the user wants and produce the expected output. It is very tedious and time consuming for this algorithm to be doing this, but it was my implementation nonetheless. 

In conclusion, my implementation runs at a sufficient speed, but ultimately with some changes in how I allocate and store my large structures, I think I could of produced some faster Ackermann values.