Daniel Frias

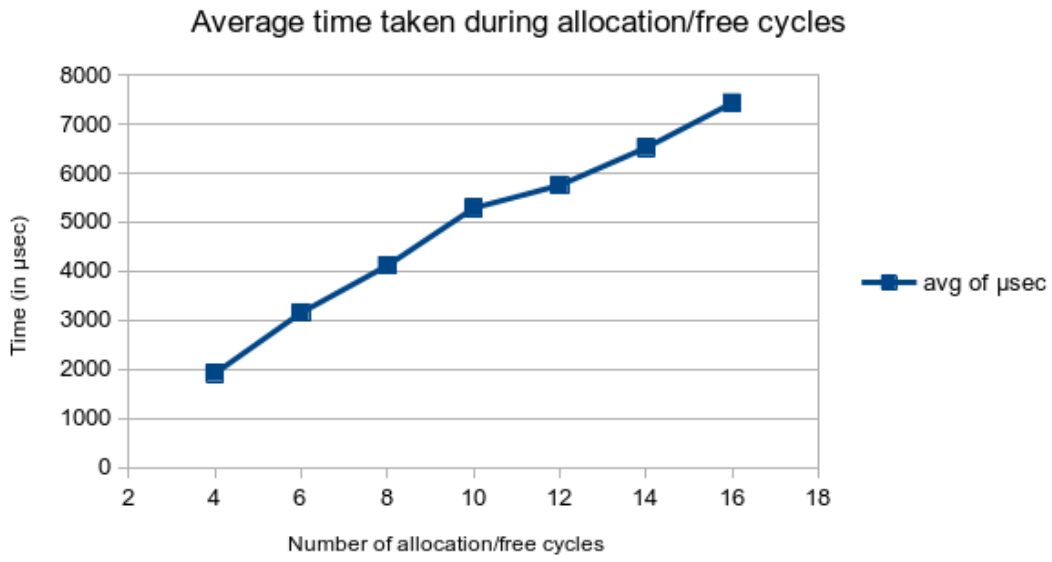
September 20, 2020

CSCE 313

Section 516

Machine Problem 1 Analysis

We begin the performance analysis by varying the values of *a* and *b,* running the ackermann functions five times with those exact same values, then we plot the **average** time taken during the allocation/free cycles with respect to the number of cycles that the program went through. Here is the graph that shows the relationship between the time taken and the number of cycles:



From the graph, we can see that the time taken with respect to the number of cycles is more or less linear. We can conclude that the implementation of this system of memory allocation is pretty efficient for being such a simple allocator. One bottleneck that can be seen in the code is how the fibonacci function runs pretty much every Malloc() call, if the input of the function did not result in a return value of either 1 or 2, then the function has to compute the fibonacci number every single time. One potential improvement may be to hold an array and compute all the fibonacci numbers it needs or maybe the first ten or so fibonacci numbers at the time of construction, this would allow the fibonacci

function to just look up the needed value in the array instead of computing it every call. Another bottleneck in my implementation is that the SegmentHeader object does not store the fibonacci index of the segment in it, as a result, the inverse fibonacci function has to compute the index every time Free() is called. The obvious solution is to just store the fibonacci index in the segment header so that the inverse fibonacci number does not need to be computed, though if the previous improvement were to be implemented then this is not much of an issue. In conclusion, the implementation of the buddy-system allocator is efficient, but there are two bottlenecks holding it back from performing even better which include the implementation of the fibonacci function and the Free() function.