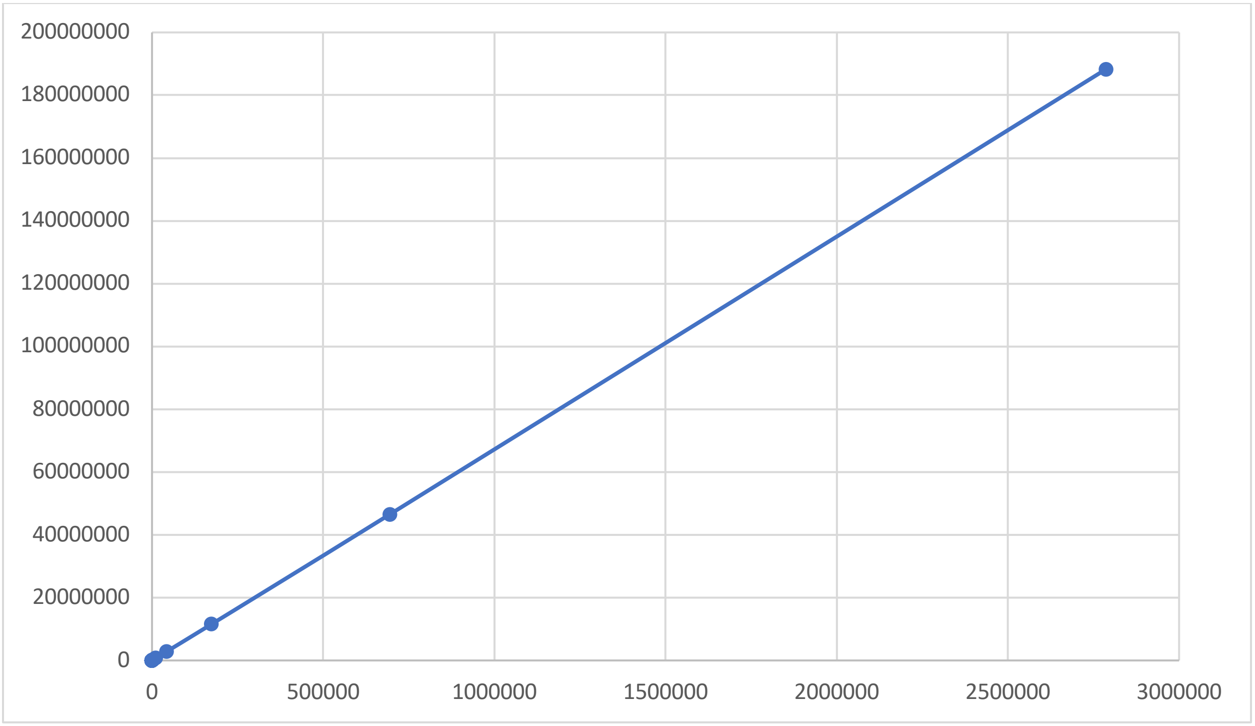
Code Performance Analysis PA1
Jiarui Li

Upon doing this project, I was able to understand the concept and implementation of this easy and powerful Buddy Allocator. Its goal is to minimize the calls to system memory allocations while managing the already allocated memory by giving the user the memory blocks with correct sizes.

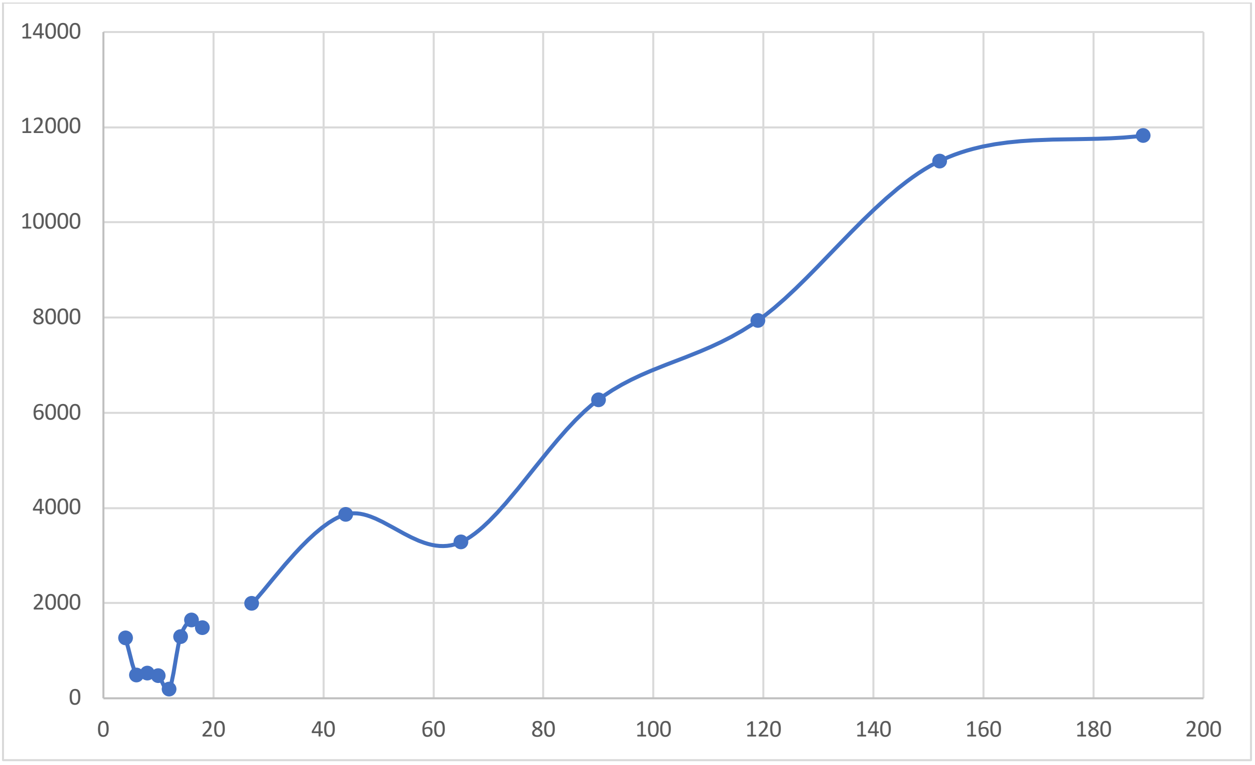
After completing all the implementation, I began testing my code upon the Ackerman function. I tested every combination of n and m – n ranging from 1 through 3 and m ranging from 1 through 8 – and recorded their number of allocate/free circles along with the time that takes to finish the circles in musec. I then put the recorded tuples into a excel spreadsheet and conducted two graphs with different sets of data.

Below the graph is conducted using all the 24 tuples. The horizontal axis denotes cycles and vertical axis denotes time in musec.



According to the above graph, one can conclude that overall, the program’s allocation/free cycles and runtime are of linear relation. But the information provided in this graph is limited because as the Ackerman’s n and m number go up, the gap between each data set is becoming bigger. That makes the graph too dense on the left and too scattered on the right, which makes one hard to tell the real behavior of the program when there are not so many allocation/free cycles.

That’s why I conducted another graph that only uses the small value of cycles and their corresponding time. Again, the horizontal axis denotes number of cycles and vertical axis denotes the runtime in musec.



According to the graph, within 20 cycles, the runtime is fluctuating and not showing an explicit relationship between both axes. But observation can be made that the runtime for small numbers of cycles is low and all below 2000 musec. Even though the relationship is not stable, one can still conclude that the program runs with stable quick speed with small numbers of cycles. What’s more, there’s a minimum value at round 15 cycles and after the min, runtime is starting to show a positive proportional relationship to the allocate/free cycle number, though with a few up-and-downs along the graph. I could not go straight ahead and conclude that the bottleneck of the program is at 15 cycles because there are so few data to be analyzed and 2000 musec is such a short runtime that may not reflect the real relationship between data. I can only say that the program is not at its stable stage before 15 cycles.

To conclude, the runtime is positively proportional to allocate/free cycles and the system starts to behave stable after around 15 cycles.

After analyzing my code, I found all the loops and functions are necessary and not redundant with the runtime of O(n) at most. I don’t find anywhere that I could modify my implementation that could result in a significant runtime boost. But outside the rule of this Buddy Allocator, we can use a other data structures like binary tree or heap to maintain the free blocks and split blocks instead of the vector of linked lists since this different logic may decrease the need of loops that traverse through the list thus improving the runtime performance.

Thanks for grading my assignment! Best wishes!