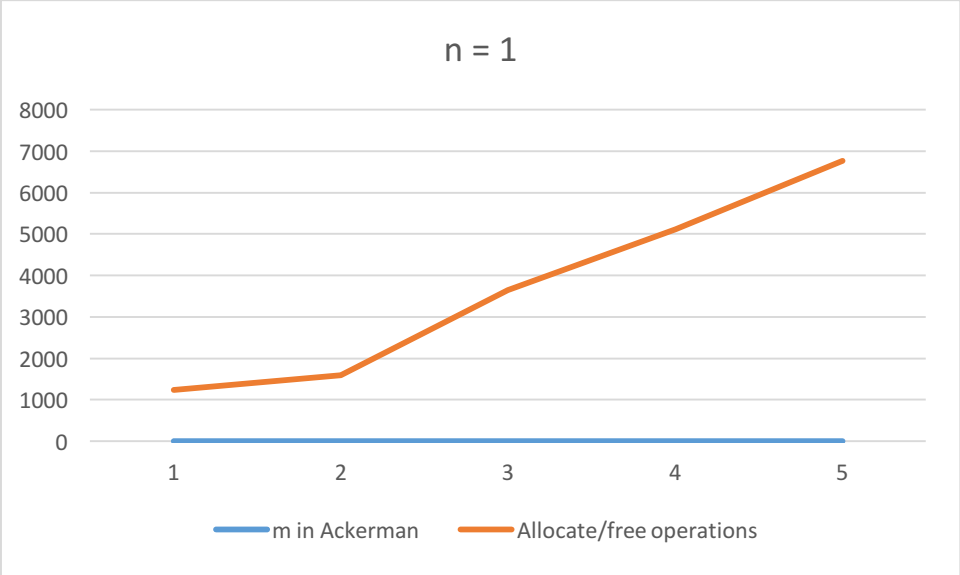
Duy Nguyen, Neal Patel

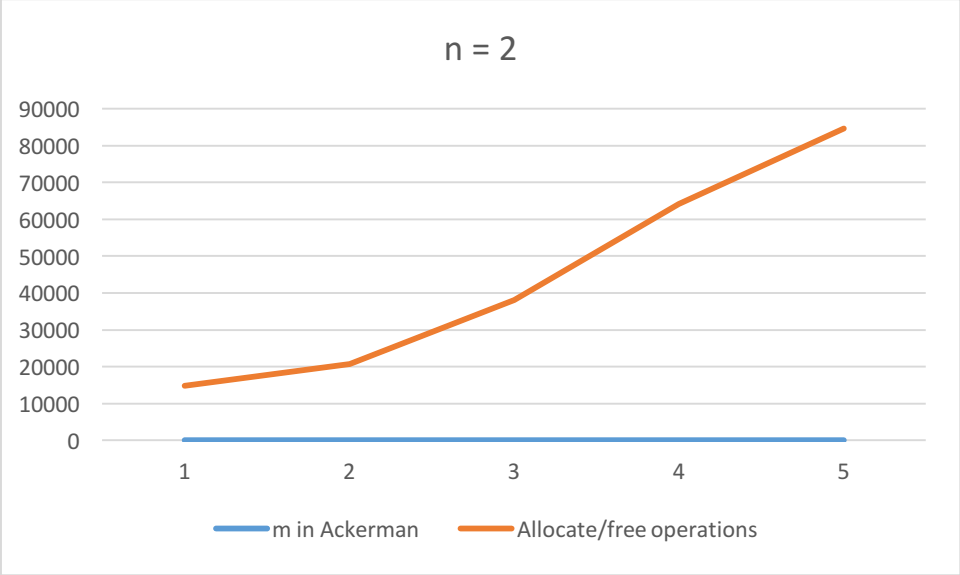
CSCE 313 – 503

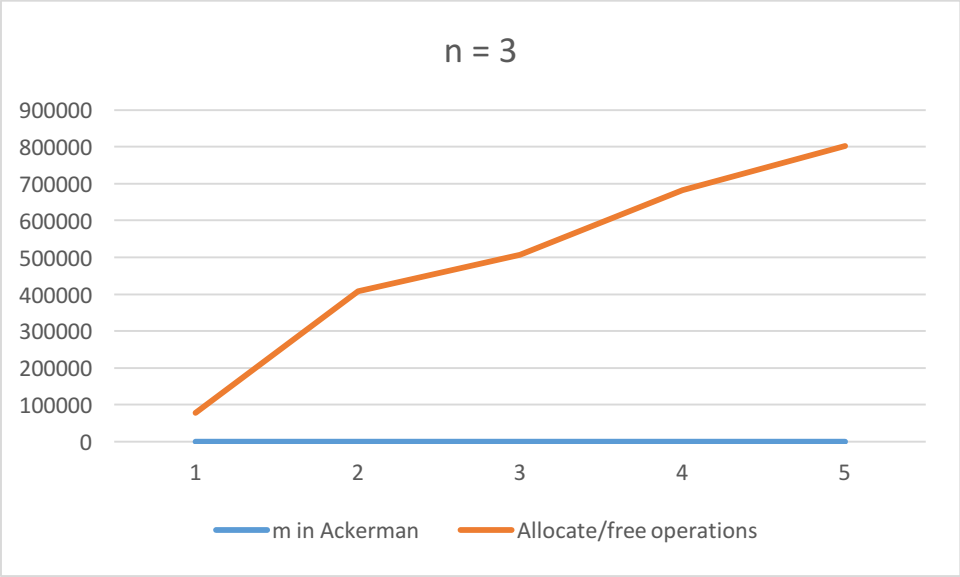
September 23, 2016

Machine Problem 2 Report

The performance of our allocator varies based on the values of m and n in the Ackerman function. Here is an idea of how the values affect the performance.







When dealing with memory management, implementation is key. Our allocator suffers in terms of performance, precision, and efficiency when the my_free function is called. Our function isn't completely freeing all the memory. For instance, when Ack(3,3) is called before Ack(3,4), then it works. And if Ack(3,4) is called before Ack(3,5), then Ack(3,5) works. However, if we jump straight into Ack(3,5), it will not work. This error implies that all the memory isn't completely being freed after each run; meaning that some memory is still being left over even though no errors are showing in previous executions. If our implementation were to completely free all the memory, the Ackerman function would work with any m and n. If we had more time, we would implement a buddy bitmap in each free list in order to properly indicate which blocks are free and which are allocated, which would additionally help us find the buddies more efficiently. An actual bottleneck would be making our slipt_block() function fully recursive. As of now, our implementation of it is only recursive when a specific free list is NULL. If we were to expand this recursive-ness to the else condition as well, somehow, the performance of our allocator would increase. Our merge_block could also use a touch of recursive-ness when

merging the blocks together instead of accomplishing it through a range until no more blocks are eligible to be merged; it would save an immense amount of time and increase performance as well.