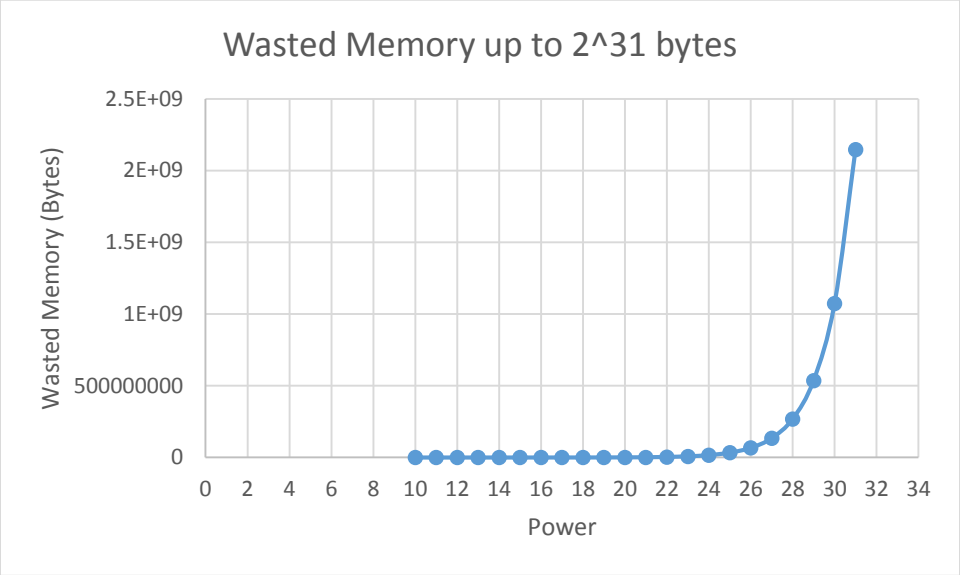
Analysis of Performance

When running the program it is fast for the small values of n and m but mainly when n is smaller but when n is raised to 3 and m is above 3 then the program runs for a little while. We tested it with a memory size of 1048576 or 2^{20}, and a block size of 8. We think that most of the problem is with the wastage this system can create. Say if the user enters the memory size of $2^{20}+1$, our system the way it is created goes up to 2^{21} which is wasting a ton of memory space. The reason we do this is because it makes it easier for the split function to get down to the basic block size. One thing that we would change in this buddy system is the fact that it just goes up to the next power of 2 because with a huge memory size that gets requested is barely above the last power of two then it has to go up and wastes a huge chunk of memory. This could be changed by splitting and bringing the memory blocks together a different way and instead of only using powers of 2 for larger memory sizes, you could check to see if it is going to waste a lot of space and if it is then you could call a different allocation function that would take a smaller chunk of memory but still make the buddy system easy to work with.



The graph above is showing that if you ask for a block of $2^{29}+1$ then it allocates 2^{30} bytes and waste $2^{30}-(2^{29}+1)$ which is 536870911 bytes and it exponentially grows from that as shown in the graph. That's an impractical amount of memory allocation but it shows how it could get out of hand real fast, and also shows what a waste this kind of a buddy system is. So out of everything we would change that because that seems like the biggest problem, but making this improvement would make it a lot more complicated throughout the program.