Eric Gonzalez
Troy Edwards
02/29/2016

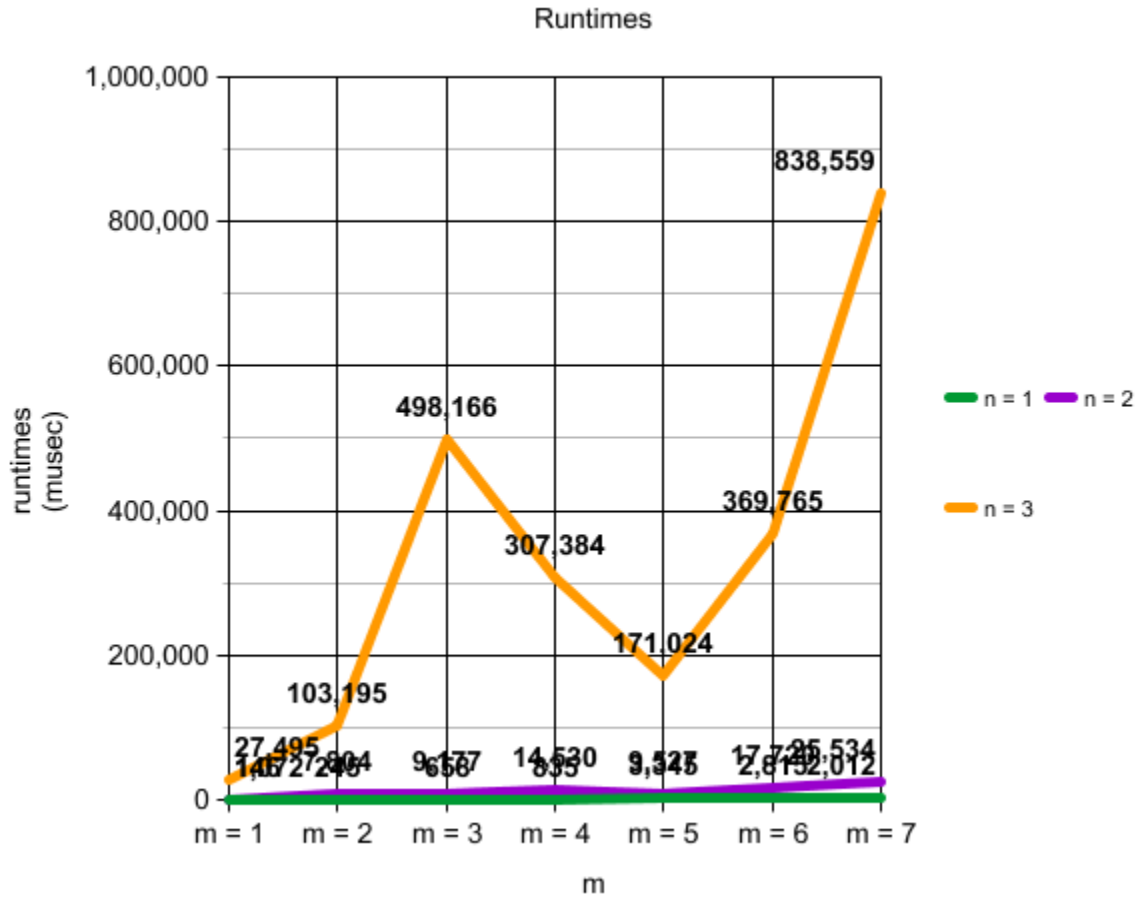
Machine Problem 2 Report

Our implementation of the memory allocator for machine problem 2 was tested using the default block size of 128 and a memory of 100,000,000 bytes to ensure there being enough space for larger Ackermann function values. The results can be seen in the table below.

N	M	Return Value	Runtime (sec/musec)	Allocations
1	1	3	0/146	4
1	2	4	0/245	6
1	3	5	0/658	8
1	4	6	0/835	10
1	5	7	0/3345	12
1	6	8	0/2815	14
1	7	9	0/2012	16
2	1	5	0/1072	14
2	2	7	0/7804	27
2	3	9	0/9177	44
2	4	11	0/14530	65
2	5	13	0/9527	90
2	6	15	0/17720	119
2	7	17	0/25534	152
3	1	13	0/27495	106
3	2	29	0/103195	541
3	3	61	0/498166	2432
3	4	125	2/307384	10307
3	5	253	10/171024	42438
3	6	509	43/369765	172233
3	7	1021	204/838559	693964

Our results proved correct, as we were able to get appropriate results for all Ackermann values when sufficient memory was provided to the program. Runtime results in musec time can be seen in the following graph:

Eric Gonzalez
Troy Edwards
02/29/2016



The program's "bottle necks" are without a doubt when large amounts of memory are initialized with small block sizes and when small memory amounts are requested from the system; because of the free blocks' large size, time has to be spent breaking the blocks down into smaller block sizes. Memory management efficiency is also greater for larger block sizes, since every block has a small constant size added for Headers that doesn't change with the size of the block. The most significant bottlenecks in implementation are in joining blocks, as it will check for the possibility even when they cannot be joined. The program does this every time a block is freed as opposed to only doing so when an allocated block needs to be split/broken.

The slowest part of the current implementation is the freeing and joining of blocks. A possible solution is to rewrite the code so blocks are only joined when there's no memory available of that block size to allocate, so that all breaking and joining of blocks are done in the allocate function. The benefit of this would be that the program could skip joining blocks for every free; however, without specific blocks being freed, the act of finding "buddies" to join would be a lot harder to implement, as well as a more expensive operation. For this algorithmic implementation to be efficient, it would mean the program having a rare need of joining blocks. So, a program that consistently allocates and frees blocks from the smallest block size would theoretically only need to break/split blocks, not join them.