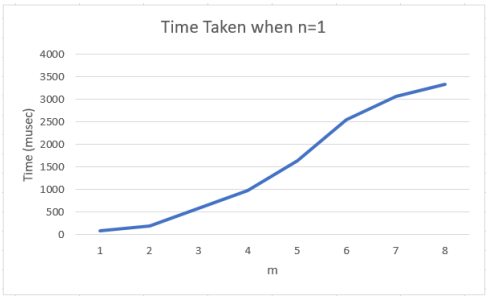
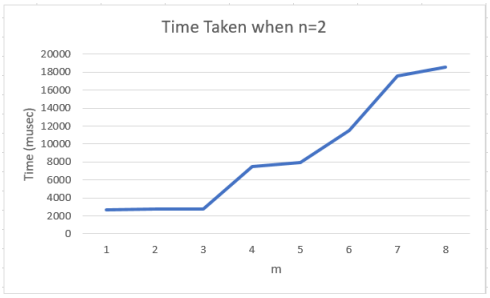
Analysis Report

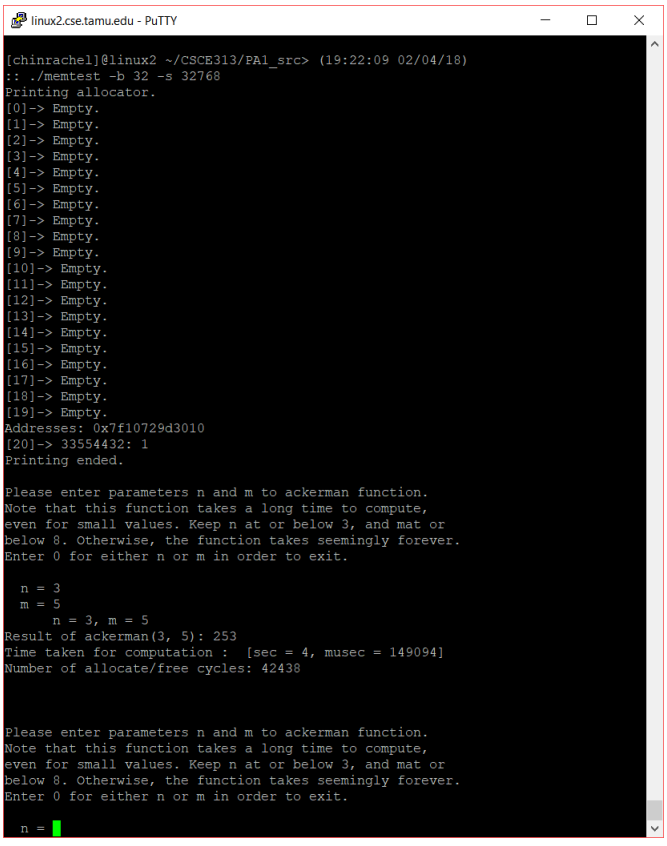
February 8, 2018

Results:

n	m	Result	Time Taken for Computation	Number of Allocate/Free Cycles
1	1	3	89 musec	4
1	2	4	192 musec	6
1	3	5	597 musec	8
1	4	6	982 musec	10
1	5	7	1648 musec	12
1	6	8	2548 musec	14
1	7	9	3061 musec	16
1	8	10	3338 musec	18
2	1	5	2711 musec	14
2	2	7	2737 musec	27
2	3	9	2787 musec	44
2	4	11	7463 musec	65
2	5	13	7946 musec	90
2	6	15	11508 musec	119
2	7	17	17609 musec	152
2	8	19	18539 musec	189
3	1	13	8786 musec	106
3	2	29	45492 musec	541
3	3	61	214059 musec	2432
3	4	125	974977 musec	10307
3	5	253	4 sec 149094 musec	42438


Time Taken when n=1
Time (musec)
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6
7
8
m


Time Taken when n=2
Time (musec)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
m


linux2.cse.tamu.edu - PuTTY
[chinrachel]@linux2 ~/CSCE313/PA1_src> (19:22:09 02/04/18)
:: ./memtest -b 32 -s 32768
Printing allocator.
[0]-> Empty.
[1]-> Empty.
[2]-> Empty.
[3]-> Empty.
[4]-> Empty.
[5]-> Empty.
[6]-> Empty.
[7]-> Empty.
[8]-> Empty.
[9]-> Empty.
[10]-> Empty.
[11]-> Empty.
[12]-> Empty.
[13]-> Empty.
[14]-> Empty.
[15]-> Empty.
[16]-> Empty.
[17]-> Empty.
[18]-> Empty.
[19]-> Empty.
Addresses: 0x7f10729d3010
[20]-> 33554432: 1
Printing ended.
Please enter parameters n and m to ackerman function.
Note that this function takes a long time to compute,
even for small values. Keep n at or below 3, and mat or
below 8. Otherwise, the function takes seemingly forever.
Enter 0 for either n or m in order to exit.
n = 3
m = 5
n = 3, m = 5
Result of ackerman(3, 5): 253
Time taken for computation : [sec = 4, musec = 149094]
Number of allocate/free cycles: 42438
Please enter parameters n and m to ackerman function.
Note that this function takes a long time to compute,
even for small values. Keep n at or below 3, and mat or
below 8. Otherwise, the function takes seemingly forever.
Enter 0 for either n or m in order to exit.
n =

Analysis

The bottleneck in the system is when the total memory allocated is large, and many smaller sized blocks have to be allocated. If there are no available blocks of that size, my_malloc() will recursively split a block larger than it, and repeat the process on the larger block if none are available. Repeatedly splitting blocks like this thus increases computation time. An implementation that may be affecting performance would be the push() function. The function iterates through the linked list and inserts a node at the end of it. As the number of blocks in a linked list increases, the time taken for this operation also increases. One point to modify in this simple implementation to improve performance may be to have the my_free() function merge lazily. The blocks that have been freed can be re-used. When a block of a larger size is needed but there are none larger that can be split to that size, the my_free() function can then be used to merge smaller, available blocks to form the larger block. This will reduce the number of split and merge operations, and thus improve performance.