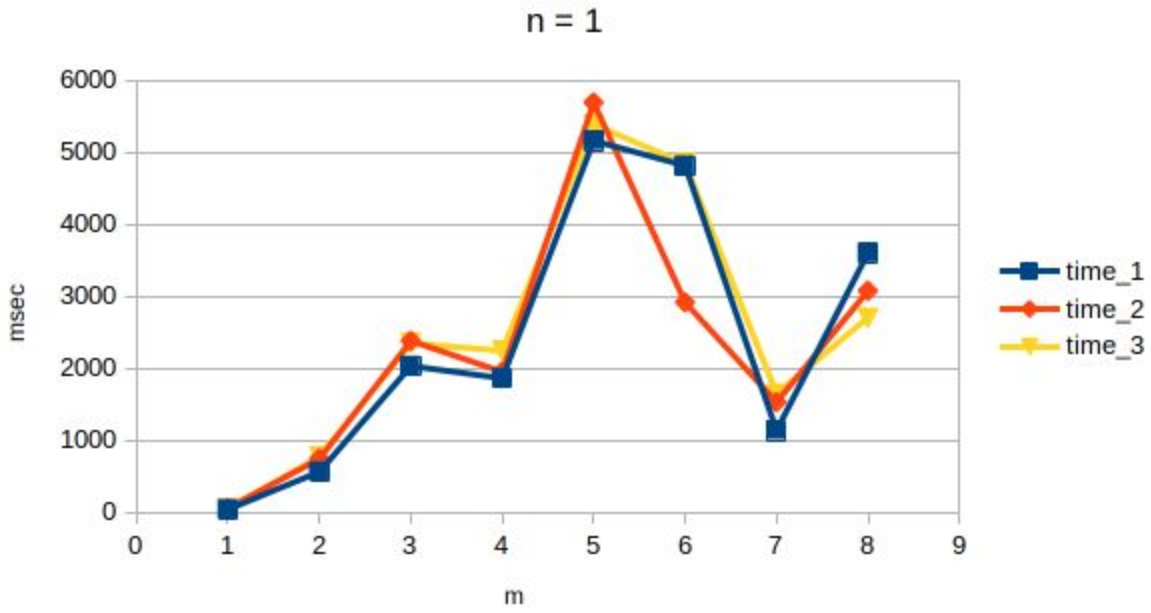
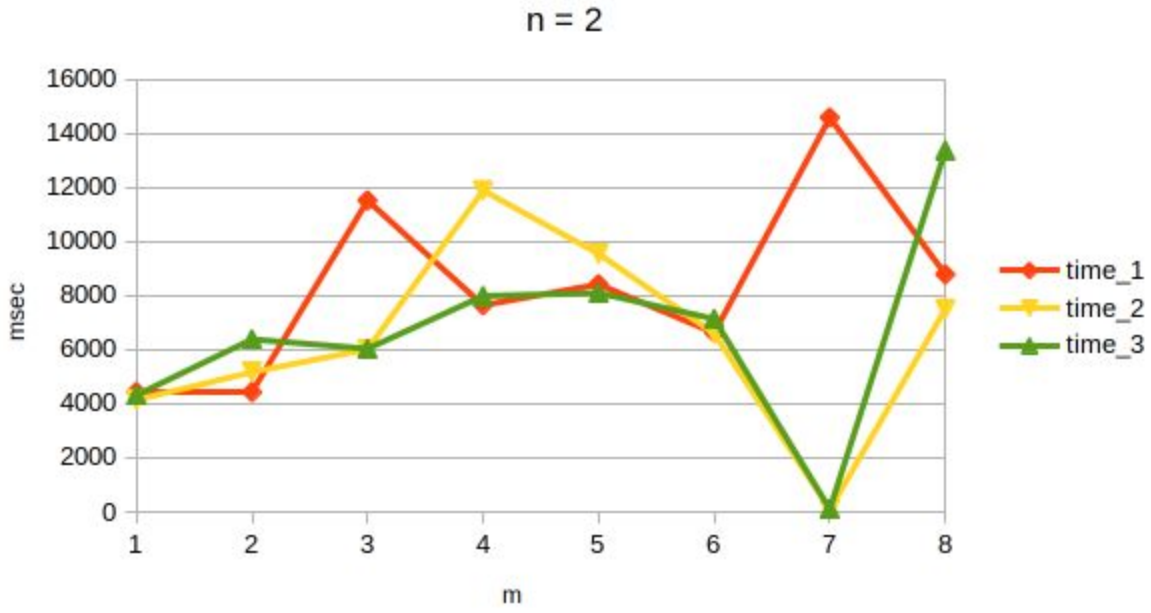
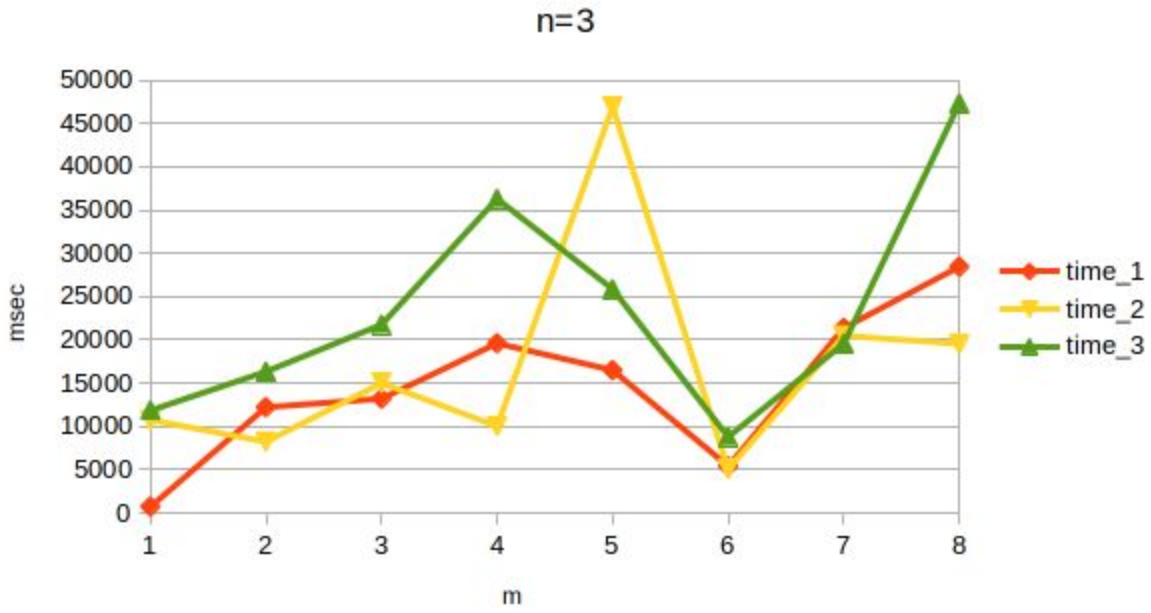

n = 1
6000
5000
4000
3000
2000
1000
0
msec
0
1
2
3
4
5
6
7
8
9
m
time_1
time_2
time_3


n = 2
16000
14000
12000
10000
8000
6000
4000
2000
0
msec
1
2
3
4
5
6
7
8
m
time_1
time_2
time_3



While the graphs vary there appears to be some consistency at regions where m is strictly less than 3, only growing exponentially as n grows larger. WIth that said though, we notice that when n less than or equal to 2 and m strictly equals 6 the graphs appear to converge closer before deviating again. From the looks of these graphs id probably chalk up these deviations to overall system performance while running the executable, as performance varies from system to system. In addition to that fact I also would be surprised to see negative impacts on performance by compiler optimizations as the code is broken down into assembly from c++. Now this is not to say my code in itself is without room for improvement, as just knowing the fact that compiler tries to optimize code to improve efficiency means that if i redesigned my code under this fact and with a greater depth of knowledge in regard to how it optimizes i could develop my code base differently, write parts where efficiency matters most in assembly, or use commands to hardware like those from ctz commands to improve performance.