CSCE 313 PA1

Christopher McGregor – 726009537

1. Performance

Performance almost directly tracks the number of Ackerman calls - fewer allocate/ free calls mean shorter runtime. With later value calls, this relationship appears to be exponential. This can likely be attributed the implementation for the doubly linked list, and iterative calls when traversing the list.

2. Performance Improvement

One point for improvement would be more efficient list creation and traversal. As the linked list is the heart of the allocator, faster traversal will decrease run times.

Another point of improvement could be only merging when absolutely necessary. In the current implementation, after each free call we check whether or not we *can* merge blocks to the highest increment. Doing this requires extra alloc() and free() calls since we constantly merge to the highest power and then split all the way down in the next request. By leaving smaller increment blocks we can drastically reduce the alloc and free calls (since both have recursive loops) – in turn improving runtimes.

3. Solution testing

I have tested all function and Ackerman combinations from (1,1) to (3,8) using the grading rubric – all of which appear to yield the correct output. Also, I demonstrated my running during lab on 9/21.

4. Data Collection

Data was collected using average run times per call (see below). Table 1 tells us how many allocate and free cycles are requested with increasing m and n. Table two averages the run times with the same calls. It is evident that both amount of cycles requested and runtime increase exponentially. This is especially prevalent when n >= 3.

5. Tables

	m							
n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	4	6	8	10	12	14	16	18
2	14	27	44	65	90	119	152	189
3	106	541	2432	10307	42438	172233	693964	2785999

Table 1: Ackerman alloc / free cycle amounts with varying m/n

	m							
n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	24	154	435	391	1263	1372	1700	1085

2	738	2932	2419	4041	6745	5675	9324	7769
3	4663	31170	126451	566972	225573 2	918066 1	3746418 2	15023555 2

Table 2: Ackerman runtime with varying m/n

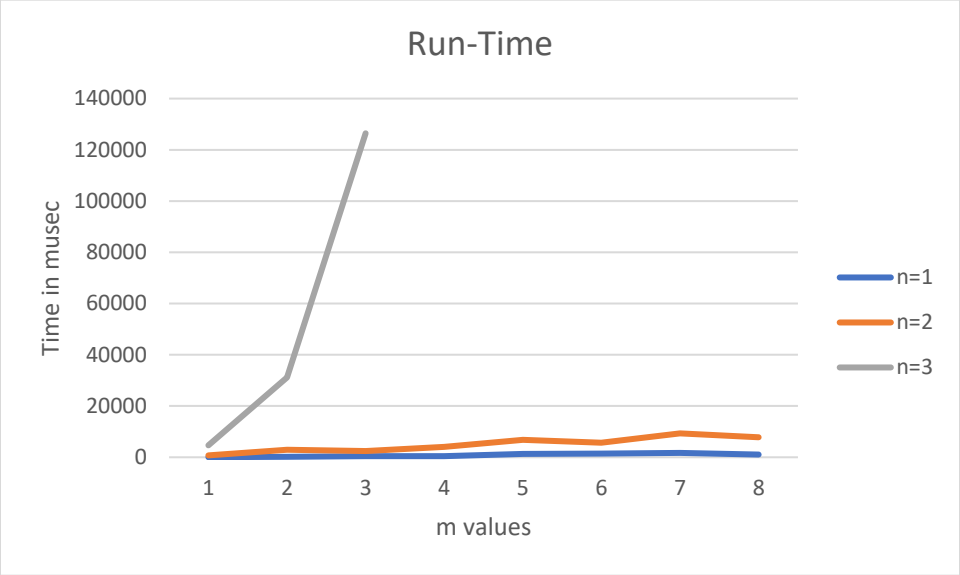

Table 3: graphical representation of Table 2