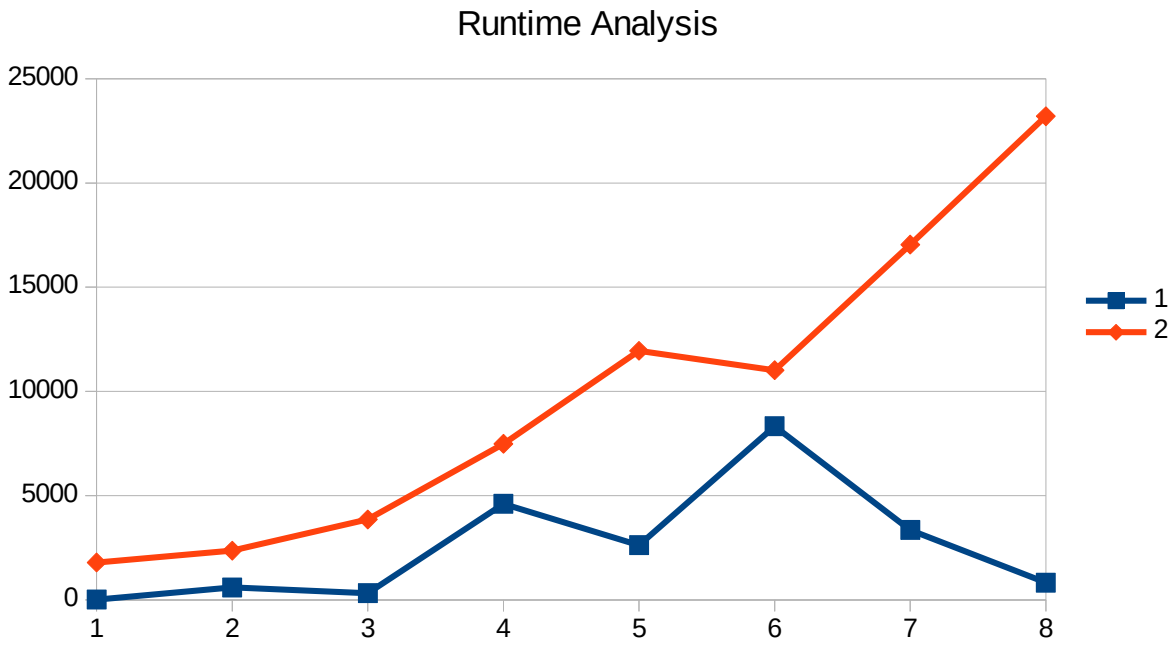
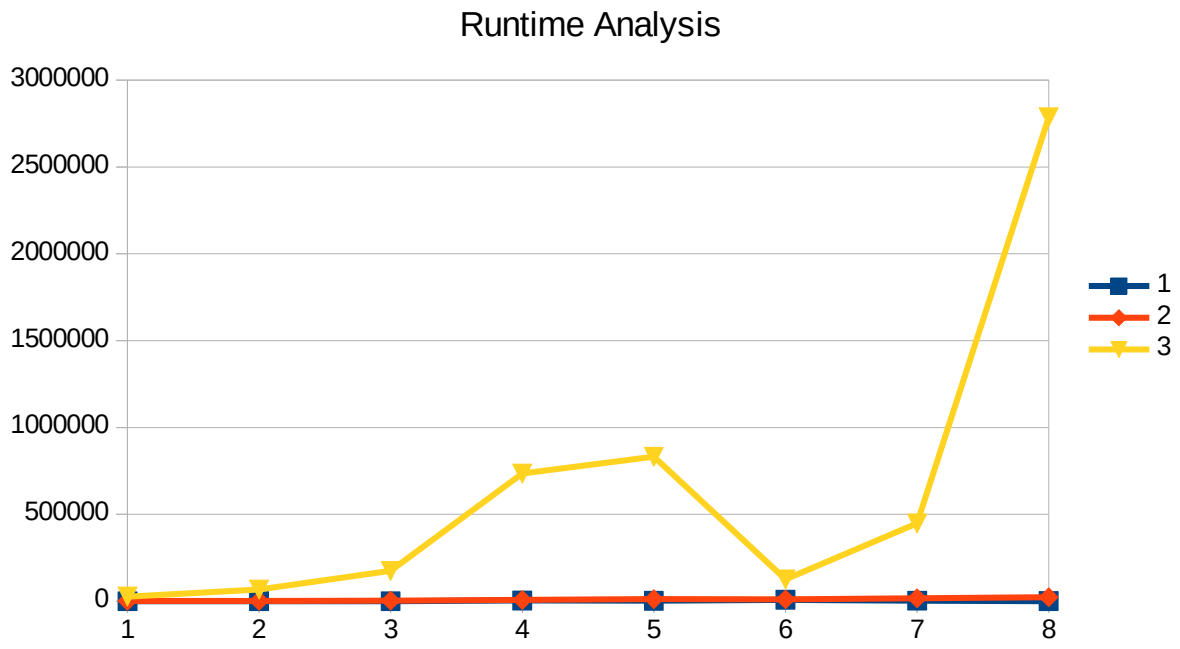
The bottlenecks in the system is coming from not being able to directly access the memory location in the linked list, thus having to traverse all the way through the list. If the memory addresses were hashed rather than put into a linked list, this would increase performance of the program. This would increase performance because traversing through a linked list takes O(n) time where n is the number of elements in the linked list and a hash table takes O(1) time to access Additionally, there is some wasted time contributed by inserting an element into a linked list after every iteration of the merge call. After the element is added to the linked list, that element will most likely have to be removed from the linked list after the next iteration. Instead of automatically inserting the element into the linked list, that block could be held until we know that we are done with merging; at which point, we then add the block to the linked list. This slowed performance is fairly negligible and does not have an asymptotic difference as is present in the previous example, but it does take extra work and will still result in slower performance.

Additionally, results are shown below for the increases in the number of allocate and free calls from the buddy memory allocator. As expected, the increased number of allocate and free calls results in reduced performance and higher overall runtimes.

Runtime Analysis									
		n							
		1	2	3	4	5	6	7	8
m	1	17	601	329	4610	2625	8334	3355	827
	2	1796	2364	3863	7487	11950	11016	17047	23201
	3	26751	67716	174940	735221	831465	125344	447713	2785999



The yellow line above represents m = 3. Red is m = 2 and blue is m = 1. The horizontal axis shows the increase in the value of n and the effects of time showed on the vertical axis.



Cycle Analysis									
		n							
		1	2	3	4	5	6	7	8
m	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