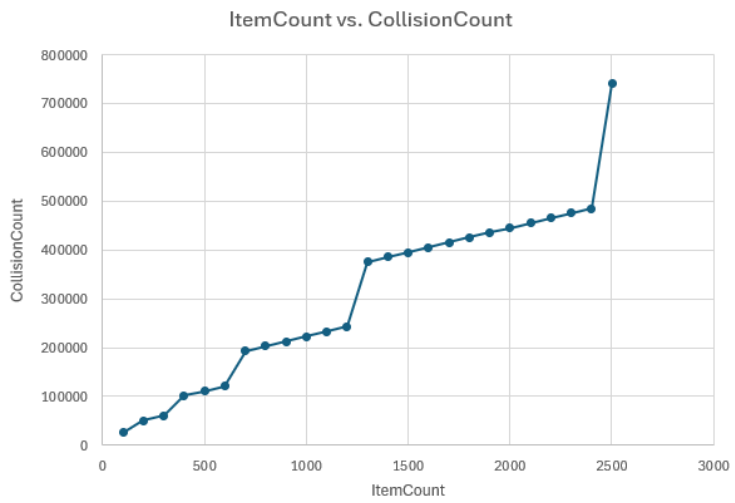
Miguel Reyes

HashMap Metrics and What Conclusions To Make

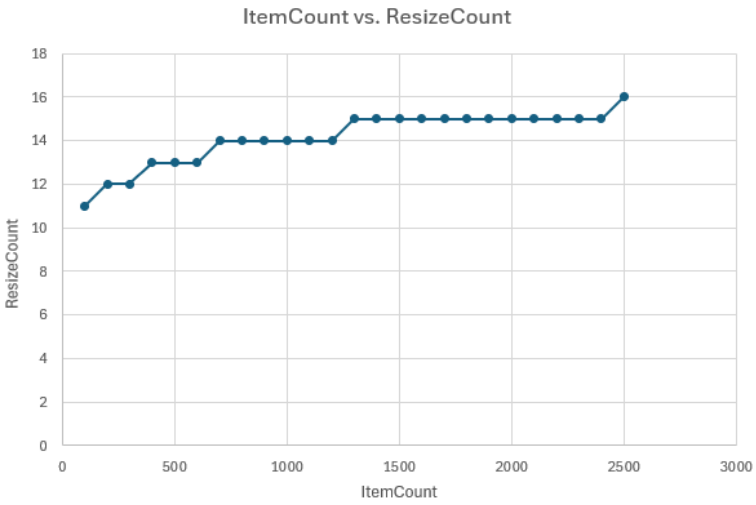
After finishing the HashMap program that deals with taking large datasets and identifying them via a key, there were interesting metrics to take note of, especially when comparing the amount of items in the hashmap versus other statistics the hashmap provides such as collisions that describes the occurrences where comparing different inputs and the value they produce which are the same. Throughout this report, I will be providing pictures of graphs and describing the findings I have discovered. Before that, I will be going over the individual metrics that were used compared to the item amount.

ItemCount	CollisionCount	ResizeCount	ElapsedTime(ms)	MemoryUsed(KB)	LoadFactor
100	25332	11	0.0066	125583	0.4883
200	50690	12	0.0068	126861	0.4883
300	60690	12	0.0091	127631	0.7324
400	101408	13	0.0058	129294	0.4883
500	111408	13	0.0058	130064	0.6104
600	121408	13	0.0061	130572	0.7324
700	192846	14	0.0031	133390	0.4272
800	202846	14	0.0031	134160	0.4883
900	212846	14	0.0031	135053	0.5493
1000	222845	14	0.0031	135438	0.6104
1100	232845	14	0.0031	136208	0.6714
1200	242845	14	0.0059	137101	0.7324
1300	375722	15	0.0031	110209	0.3967
1400	385722	15	0.0031	110849	0.4272
1500	395722	15	0.0031	111489	0.4578
1600	405722	15	0.0032	112257	0.4883
1700	415722	15	0.0031	112897	0.5188
1800	425722	15	0.0056	113537	0.5493
1900	435722	15	0.0031	114305	0.5798
2000	445722	15	0.0078	115265	0.6104
2100	455722	15	0.0036	115712	0.6409
2200	465722	15	0.0035	116673	0.6714
2300	475722	15	0.0051	117313	0.7019
2400	485722	15	0.0031	118081	0.7324
2500	741479	16	0.003	128641	0.3815

ItemCount represents the number of items in the hashmap so far in each row. The increment that was used when charting the item amount is 100 in order to explore a large set of numbers for a better look at how the hashmap works. CollisionCount is the number of times collisions occurred over time and builds off of the previous numbers since this counts the total number of collisions that occurred in this trial. The ResizeCount, which is tied to the number of times the array is doubled in size and the elements being rehashed, is the number of times the resize function had to be used to make all the elements being used fit in the Hashmap. ElapsedTime(ms) isn't built off the previous value in each column since it focuses on each specific instance of hashing and how long it took for this specific instance of hashing to execute. MemoryUsed(KB) is the amount of memory used in each specific instance of hashing being executed and is done using the measure KiloBytes, used specifically for computing. LoadFactor deals with indicating how full a hashmap is by comparing the number of elements to the total number of slots available for elements to be added to. Since this is below 1, the ratio in each column indicates how full the table is with 1 indicating a full table. Now that I have explained what each metric indicates, I will be comparing them to Item count and at times, with each other.

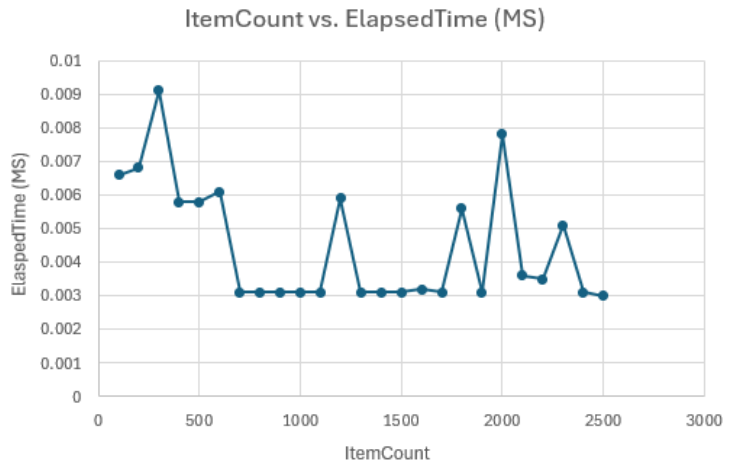


The ItemCount vs. The CollisionCount line is constantly growing when being compared to each other, following the intended logic for each metric since the count is built up throughout the hashmap program. An interesting note is the upticks that occur when reaching certain increments, indicating when collisions are happening the most in the program. Usually the points follow a close to horizontal line but if you focus on 300 for ItemCount, you can see a close to noticeable uptick for the collisions. Next, it occurs at 600 at what appears to be double the size of the collisions occurring at 300 and this pattern goes on by doubling these numbers, starting from 300. 1200 and 2400 for ItemCount go through upticks and at double the amount of the previous one. This shows that collisions are occurring at specific instances and can be predicted based on following the pattern on the current graph.

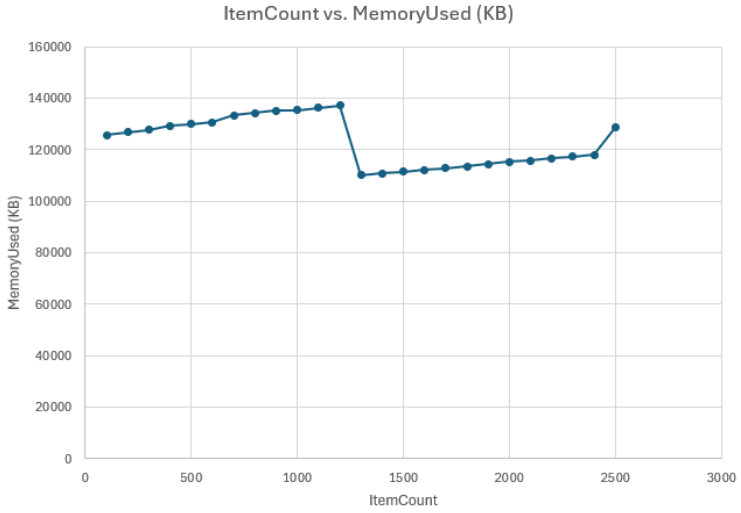


The ItemCount vs. ResizeCount is a pretty constant line since the metrics for the ResizeCount is close to constant throughout the program, indicating not a big need for the resize function throughout the program, even though 11 resize calls by 100 items is pretty high as is. The pattern of when a resize is not as noticeable but there are upticks occurring at specific instances. Usually, a resize occurs at a doubling rate from the previous amount of items that were

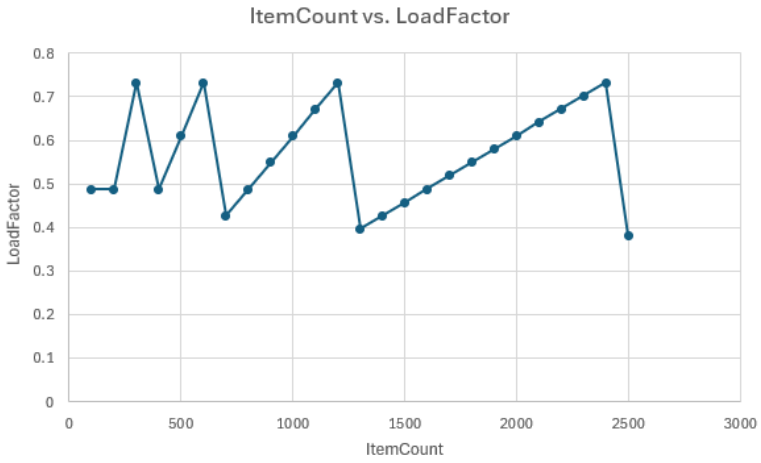
counted before a resize. Specifically, at a 400 and 600 it is at a constant resize amount and goes up one from 600-700, whereas the next resize occurs after the 700-1200 range in ItemCount which is double the previous range. The same thing occurs when looking at the 1300-2400 range, also double the previous before the resizing occurs from 2400-2500. This is also another metric that can be predicted when looking at the ItemCount compared to itself since we can predict when the next resize will occur based on the previous range and doubling it.



ItemCount vs. ElapsedTime(ms) isn't as consistent in growth compared to the previous two, mainly because of time focusing on specific instances over focusing on the count amount. An interesting thing to take note of is adding items into the hashmap won't take more than 0.003 seconds and won't be more than 0.009. The unpredictability in the time lies in a lack of a clear pattern. Along with that, adding items into the hashmap is closer to the lower bound (0.003) more than it is closer to the upper bound (0.009), showing the quickness of adding elements into hashmaps. Out of the current graphs, this one is hardest to predict where it will go next. We will see the potential unpredictability of the metrics as we go throughout this report.



ItemCount vs. MemoryUsed(KB) which looks at the items being added into the hashmap and the memory used for each instance. I would not have predicted there is less memory being used when getting to the higher values since the rule of thumb is the higher a value is the more likely a lot of memory will be used. There is a suspected correlation when it comes to values that are incremented by the value of 1200 since the only upticks or downticks appearing only happen at 1200 and 2400. It is interesting to note that usual downticks or upticks throughout the current graphs in this report all happen at 1200, making 1200 an interesting value since it is also the closest to half of the total number of items. The reason for this is unknown but the focus is all on 1200 in the ItemCount due to this finding throughout the report.



ItemCount vs. LoadFactor, or the Load Ratio, is sort of a parallel to the ItemCount vs. ResizeCount since the downticks that occur in this graph are where the upticks occur in the ResizeCount one. This could be a potential correlation to the ResizeCount and LoadFactor metrics which tells us that when a resize occurs to the array in order to hash more elements that the load factor experiences a decrease. This fits the logic of the resize method creating more space since the load factor reflects the fullness of a hashmap, which is decreased when doubling an array. After reviewing the graphs for the hashmap function, the logic behind each one holds up with the intention of the program, especially when it comes to the resize and load factor correlation. More graphs were taken that I thought looked interesting but will not be discussed in this report for the sake of focusing on the comparisons with ItemCount. They will be submitted with the other graphs just for analysis in order to see the uniqueness of each one.