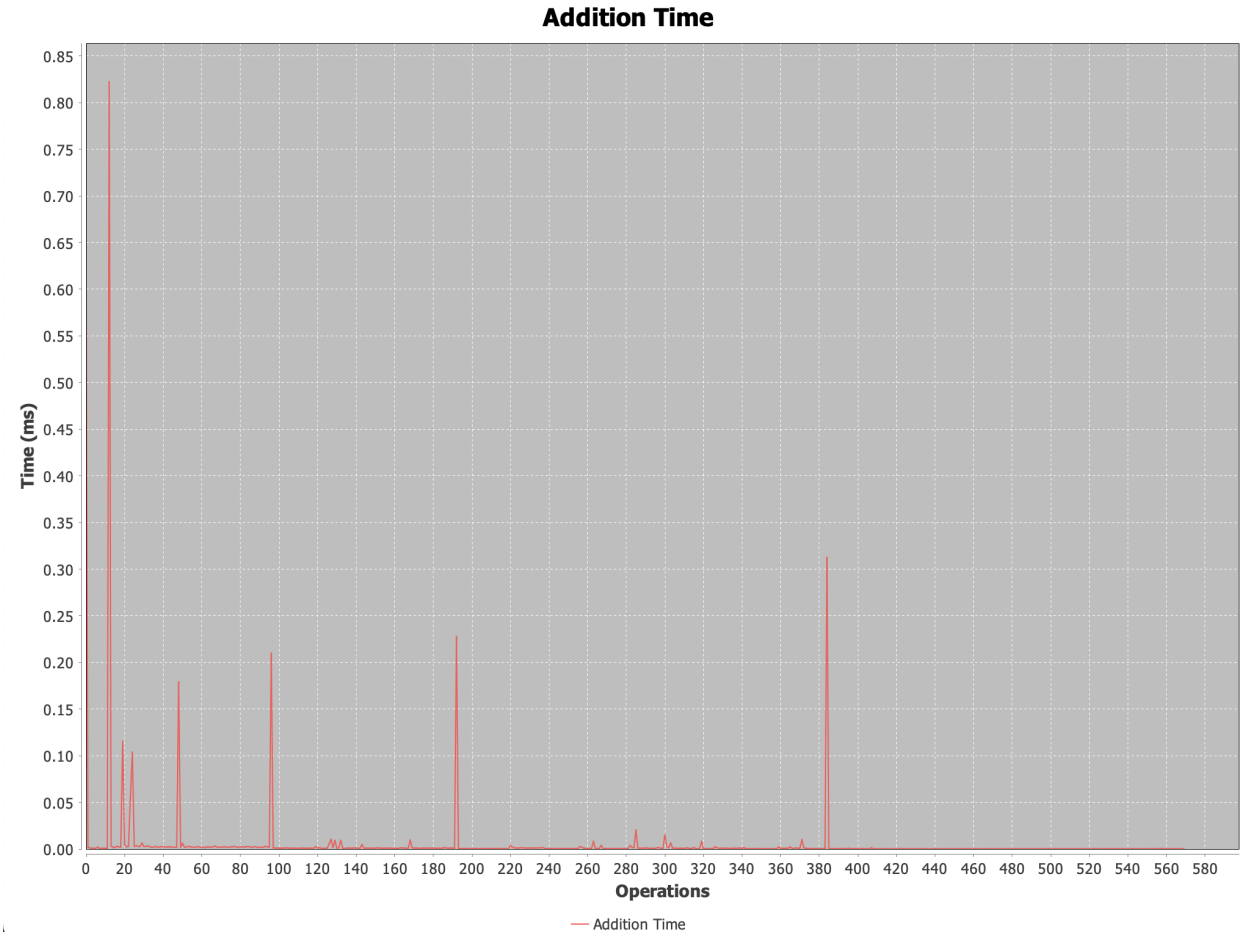
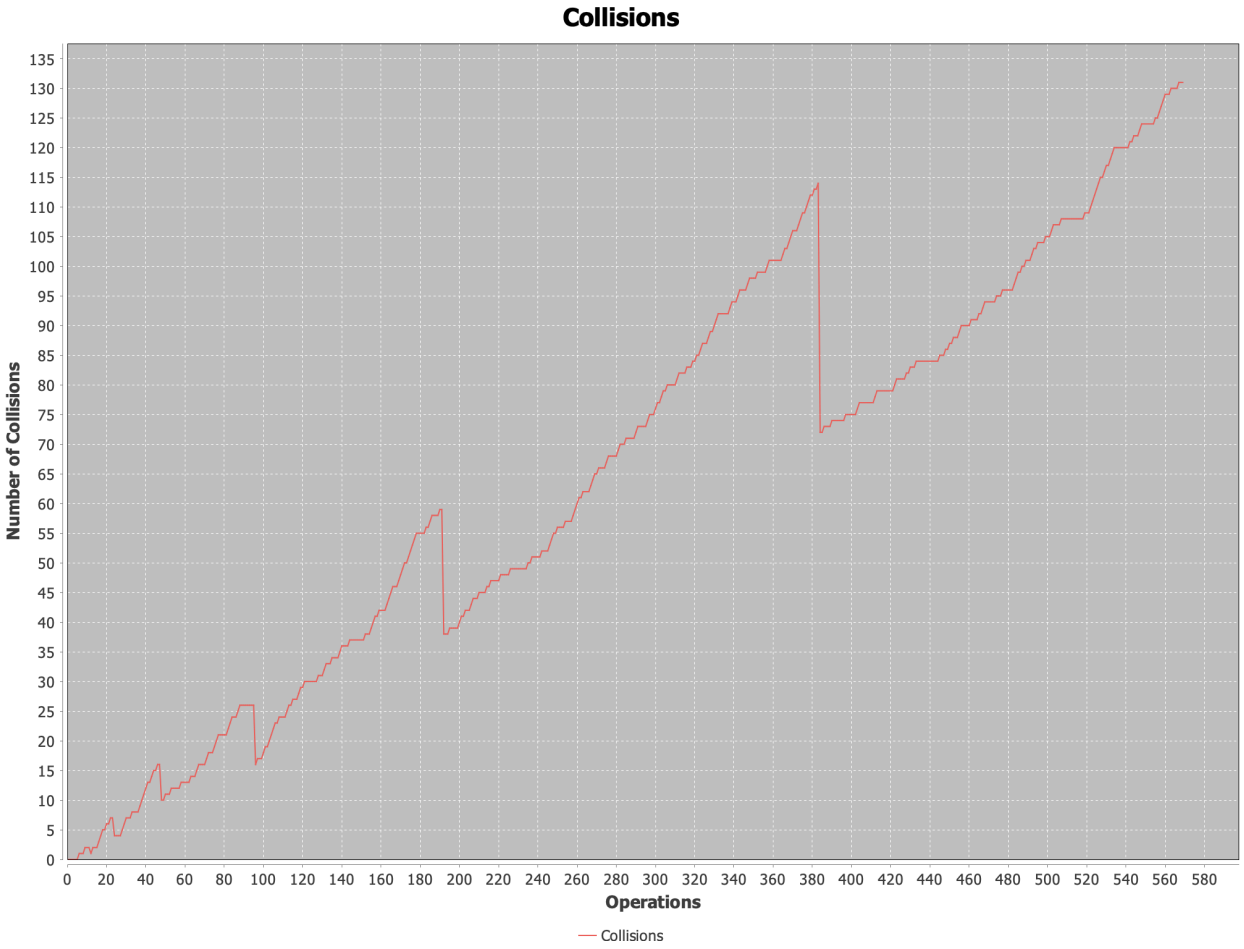
HashMap

In this write up I am testing my hash map function. I implemented my own hash map class based on the dumb hash that we did in class. Though, instead of using the dumb hash function that we made in class I changed the function so that it is a rolling hash function that uses the prime number of 31. This is similar to the implementation of Rabin-Karp that I'm currently studying in Data Structures and Algorithm 2, so it was interesting to see the overlap there. I used the data set that I used for my report but only took the names of all NBA players from it and organized them into buckets based on length with this hash function. The results were interesting and I think this hash function performed quite well compared to how the dumb hash function would have performed.This was all charted using JFreeCharts.

This graph tracks the runtime of each `add()` operation in my custom hash map. Most operations complete very quickly, but a few noticeable spikes occur — these represent resize operations. Each time the load factor exceeds 75%, the hash table doubles its size and rehashes all existing elements. This rehashing creates a temporary spike in runtime, which is clearly reflected in the graph. This behavior is expected in hash table performance and shows how resizing impacts time even if most insertions are fast.



This graph tracks the total number of hash collisions over time as more elements are inserted using the **improved hash function**. We still see a steady rise in collisions between resize events, but that's expected as the table fills up. After each resize, the collision count temporarily drops since there are more buckets available — but the climb resumes as more data is added. What's important here is that while collisions do increase, the pattern is **smoother and more gradual** than it would be with a poor hash function. The improved hashing method does a much better job spreading entries across the table, avoiding extreme clustering. This chart reflects a typical and healthy performance curve for a growing hash table — where collisions grow over time, but are **reset and managed predictably** through resizing.



This graph shows the average name length of players stored in each bucket after inserting the full dataset using the **improved hash function**. Unlike the previous dumb hash that only used

string length, this version uses a **polynomial rolling hash** based on the actual characters in the name, not just how long it is.

What we get is a distribution that's **surprisingly balanced** across buckets. Most buckets contain names averaging between 10 and 18 characters long, with only a few outliers. There are **no major spikes** or flat spots that would suggest certain buckets are overloaded or empty — which is exactly what we want from a good hash function.

This tells me that the new hash function is **doing its job well**. Even though NBA player names can have common lengths, the function mixes them enough that we don't end up with big clusters of names with the same length or characters landing in the same bucket.

If we had used the dumb hash, this graph would look way worse — you'd see a few tall bars where all the names of the same length got stuffed into the same bucket, and most others would be nearly empty. That's not happening here.