Justin Kleiber

# **Project 3 Design Document**

The hash function I chose to implement in Project 3 is a simple modulus hash function. The equation used is key % tableSize, which is the most basic hash function for a viable hash table. However, it does distribute the keys within the bounds of the hash table, which meets the requirements.

My collision resolution strategy is separate chaining using the LinkedList class from the prior project. More specifically, my table is a ResizableArray of LinkedList objects, and is a templated class that manages the chaining for the user. I believe that separate chaining was the best solution for this hash table because it keeps the data structured in an intuitive way that is easy to debug, and less likely to form clusters, while guaranteeing a worst case performance of O(n). Other functions with similar Big-O performance, such as linear probing, would be more exposed to sequences where there is clustering, resulting in a longer chain of values that do not actually match the hash being looked up. Additionally, separate chaining is easy to debug; a collision resolution strategy with any sort of probing, or a coalesced chain can result in weird side effects that are hard to correct. For example, a coalesced chain might be corrupted on resizing the hash table, and the linear and quadratic probes require a tag to indicate if buckets have been occupied before. I believe that for this project, separate chaining is the way to go to avoid these unfavorable attributes of other functions.

Since I am using separate chaining as my collision resolution strategy, all of my bucket sizes are 1, with the ability to overflow into the chains. It doesn't make sense to have larger bucket sizes than 1 for separate chaining, because the whole point of the chains is that the data can overflow outside of the small bucket in an easily searchable fashion.

My hash table is formed based on the number of input items available for insertion, as recommended by the instructions. I chose a load factor of 0.7 for my table because I felt like that was an efficient balance between minimum storage and minimum collisions. Upon testing, the most overflow for a bucket is 2 or maybe 3 eclipses, so I am satisfied with these results. Of course, specific instances where remainders are the same could potentially hurt the hash table's performance, but that lies on the hash function, not the load factor.

My hash table's embedded linked list is not the same data structure that stores everything in order by catalog number, however the two are identical upon output.