**CPSC 331**

**Calder Sloman, 30107102**

**Carlene Kalin, 10162051**

**June 2nd, 2023**

**CPSC 331 - Assignment #4**

**Hash Tables and Searching**

A picture containing text, screenshot, font, line

Description automatically generated

Figure 1

After comparing the search times that result from a sequential/iterative search to those of a binary search, it becomes immediately apparent that the binary search is much faster for any array size over 1000. This trend continues as the size of the search array – n, increases.

I would choose a sequential search anytime I was developing a program that only needed to search a dataset occasionally. The sequential search was still capable of searching 500,000 values in 0.0014834 seconds. This speed is adequate for any use case that does not depend on repetitive searches. On the other hand, if my program required the same data set to be searched repeatedly, then binary search would be the obvious choice. This brings us to the other topic that must be discussed when comparing the search algorithms. One does not require the dataset to be sorted (sequential) and the other does (binary). For this reason, binary search is best suited to a situation in which a sorted set of data is searched many times. This is due to the fact that if a dataset is sorted and then only searched once, any gains made by the efficiency of a binary search are offset by the cost incurred when sorting the dataset in the first place.

When searching smaller sets of data, or when performing a single search on a dataset a sequential search is preferable. In contrast, whenever it is likely that a dataset will be searched multiple times, sorting the data so that a binary search can be performed would likely be a better option.

A picture containing text, screenshot, plot, line

Description automatically generated

Figure 2

When comparing the results associated with searching for a value in an array with a binary search to looking for a value in a Hash Table, we can see that up until the dataset includes 100,00 or more values, binary search requires less time. After this, a hash lookup always takes less time than a binary search. Furthermore, the time required for a binary search grows as the size of n grows, whereas the time required for a hash lookup remains constant. A similar trend emerges when comparing the time it takes to sort a dataset that has elements added to it, to the time it takes to add those same elements to a hash table. In my program, I created a static set of values that reside in a text file which is then used to add elements to the search array as its size (n) increases. This means that both the array and the hash table do not have to be reset and repopulated for each size increase (iteration of the loop) – they only need to be added to. This revealed a dramatic difference between the efficiency of the two search techniques, as both the hash table and array needed to have elements added to them, but only the array needed to be sorted in order for the search methods to work. This meant that as the number of elements in the array increased, so did the time required to sort it, whereas the time required to add elements to the hash table remained constant. This made the hash table a far faster solution as n increased. The results of this comparison can be seen in Figure 3 below.

If I was creating a program that needed to retrieve information from a static dataset, i.e. no data would be added or removed, and it had less than 150,000 elements, I would definitely consider using a sort and binary search approach. However, if I knew that I would be modifying a dataset by adding or removing elements from it, and then performing a search, I would choose to use a hash table.

A picture containing text, screenshot, plot, line

Description automatically generated

Figure 3