Konan Le

Professor Long

CSE 13S

5 December 2021

Assignment 7 - The Great Firewall of Santa Cruz

Abstract:

This assignment involved implementing a hash table and a bloom filter to act as a filter for censored words as specified by the user of the program. Overall, it would make sense if the more entries that are available in the bloom filter, the lower the lookup time. there would be because of the fact that there would be more entries available for the hash to fill. Less collisions means that there is less lookup time because you decreasingly have the need to double check for false positives. If you were to vary the hash table, the height of the binary search trees decrease significantly due to having more slots for data to be stored. On the other hand, varying the bloom filter would not result in any change to the average height of the binary search tree, because the bloom filter is a completely independent function from the hash table. We simply use the bloom filter to see if the word is possibly a part of the collection of bad words - that is why in the program we add the forbidden words to both the hash table and the bloom filter.

Introduction:

In the field of computer cryptography, the use of hash functions, tables and bloom filters are synonymous with the term. With the increasing sensitivity of the data that the modern populace uploads onto the internet, it is imperative that one understands how their privacy is protected and stored, whether just to understand or perhaps even create an iteration of it later.

Much of our modern encryption comes from these abstract data structures, and by creating these data structures from the ground up, one can understand just how one-way these hashes are.

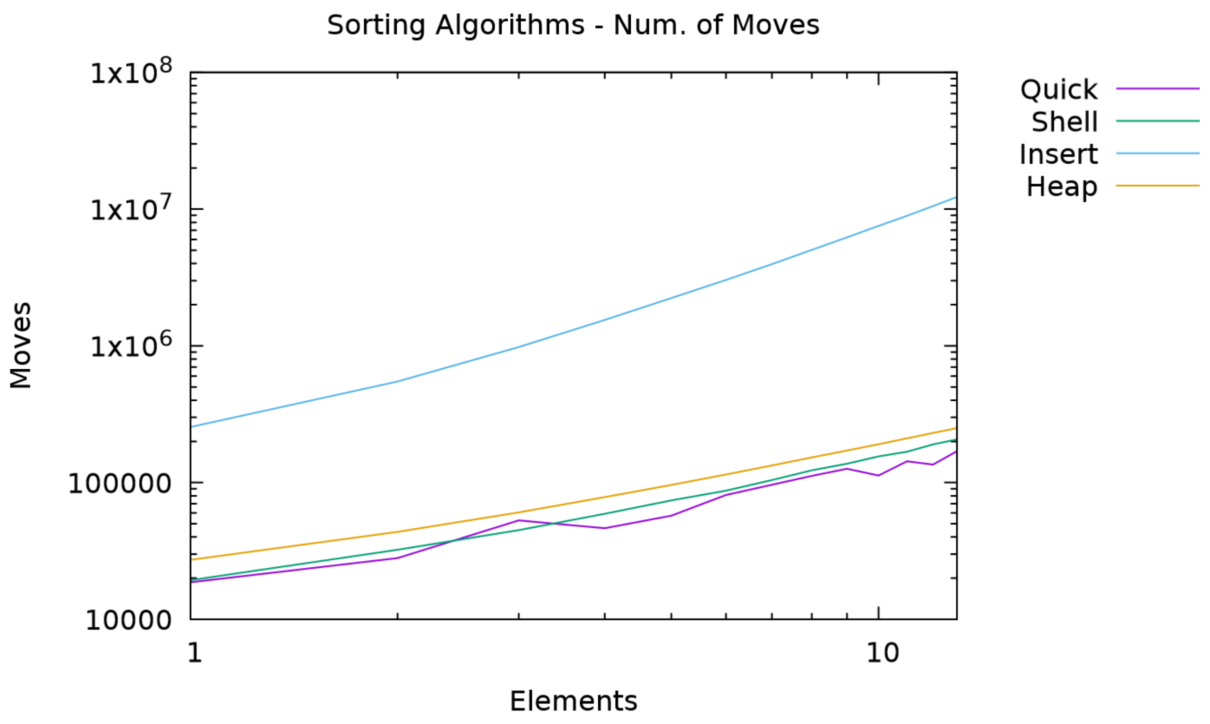
Materials:

In order to analyze the data, we need an Ubuntu Server 20.04 LTS operating system. The PuTTY SSH client was used to access the command line interface of the Ubuntu Server, which is hosted on a virtual machine. The application gnuplot was used to graph the data output from the program. Git was used to maintain version control.

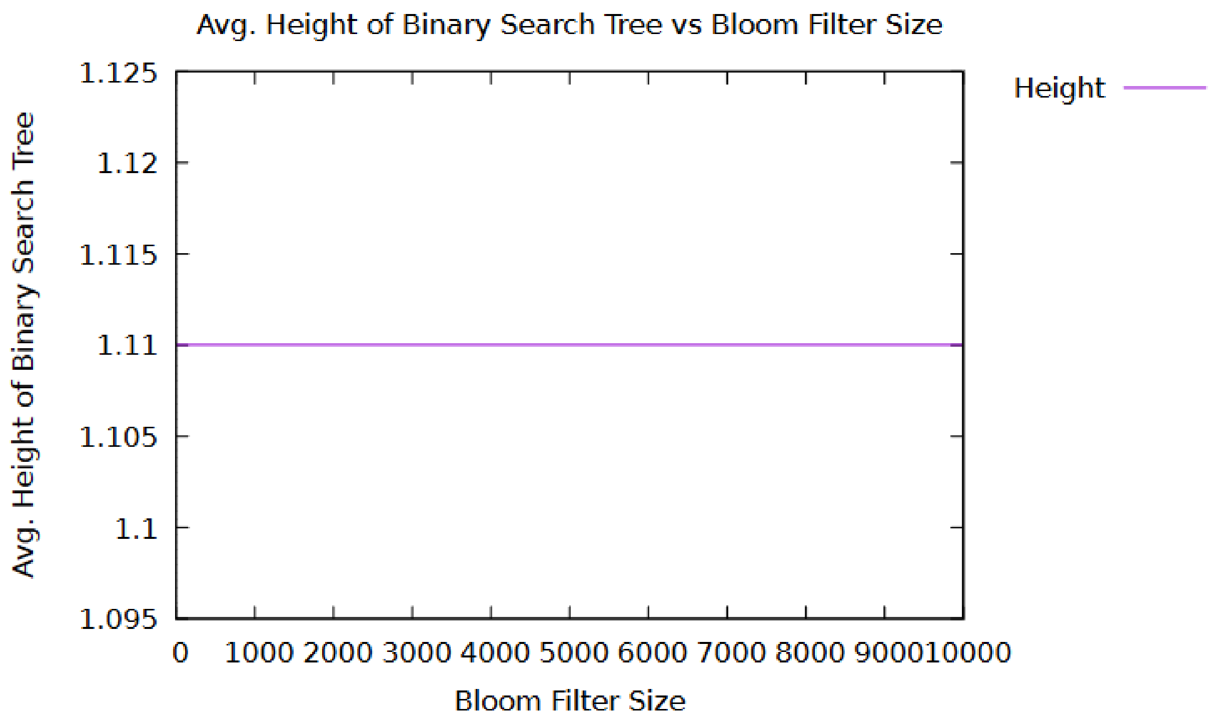
Methods:

The bloom filter and hash tablet ADTs were built from the ground up in C. We started by making bit vectors and node data structures, which built up to binary search trees, hash tables and bloom filters. The hash table data structure has a function that utilizes the ability to check the height of a binary search tree to find the average height of every binary search tree in the hash function. By varying the number of entries of the bloom filter and hash table, we can build a graph that compares how the height and overall construction (lookups and branches) of the binary search trees change. In order to test the functionality, the following input was given: “One day, I was walking along the UCSC road. Then, I met my cosin and shared a snack with him. It was then that I decided to be a curer, but certain circumstances forced me to the life of a criminal. I had to go and film one of the best police around. It took a while before I had him in my graspe”. This paragraph contains words of all different classifications - badspeak, oldspeak with newspeak translations, and words that were not badspeak or oldspeak.

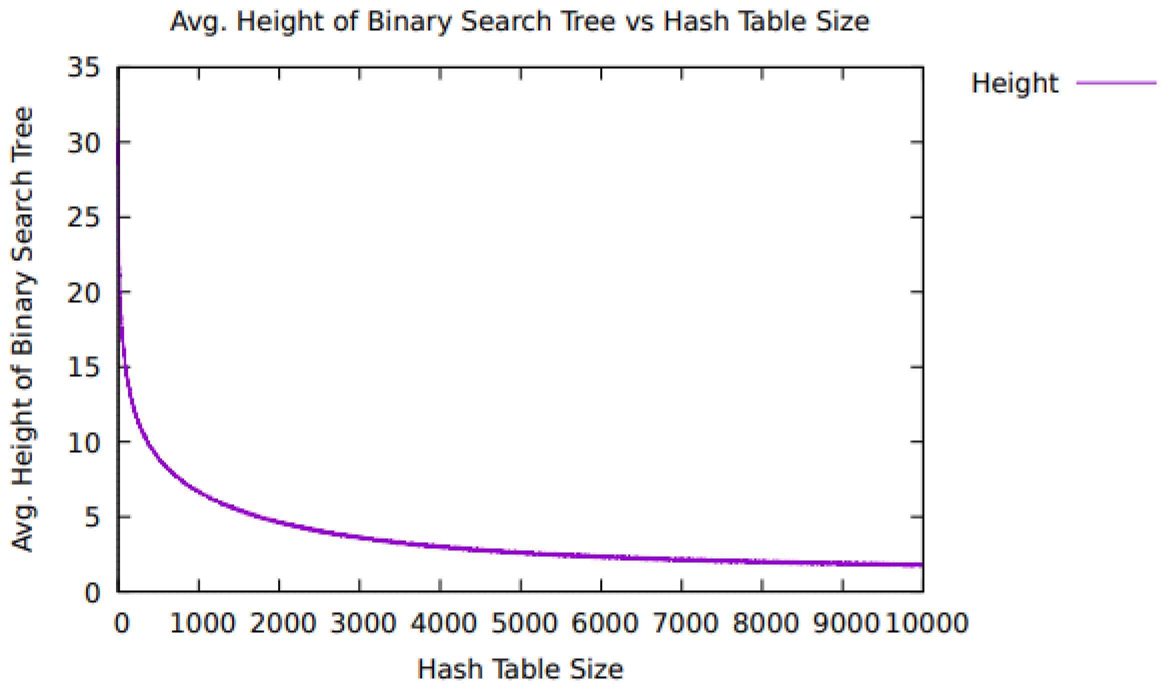
Results:



According to the graph, Insert sort takes many more moves than the other sorts. The three other sorts are relatively close to each other when sorting over this large scale set of data, not because they are faster but most likely because Insertion Sort is much less efficient. Quick sort has fluctuations, and that is due to the nature of its implementation, with the sort approaching from both sides of the array inconsistently (contrasted to the Shell sort, which also approaches both sides, but uses a consistent gap value.)



According to the graph, the average height of the binary search tree of the hash table remains constant, even after changing the size of the bloom filter from 1 all the way to 10,000. The average height calculation occurs multiple times because of the iteration occurring as the entries in the bloom filter increase, yet it remains the same. If the graph took a smaller sample size, there still would have been no visible fluctuations due to the nature of the binary search trees not relying on the bloom filter.



This graph shows that as the number of entries available in the hash table increases, the smaller the height of the binary search tree. If the sample size was smaller here, the slight fluctuations would be more easily visible. This is easily explained due to the nature of the inserting of binary search trees varying depending on the input, but in general it will remain the same. The reason it exponentially decreases but seems to stop decreasing below a certain point is because eventually, the entries and the roots of trees stored in the hash table will be spread so thin that it will take up a constant amount of space.

The graph of the average branch traversals compared to a size-changing Bloom Filter would be similar to the graph above - it would get exponentially smaller, eventually coming to a stop at a very small number, possibly 1. Theoretically, this is because the more space there is for a bloom filter to store data, the less likely it would be for a hash function to give the same hash as previous data entries - lessening the false positives. Similarly, the smaller the bloom filter, the higher the number of lookups due to the need to verify the false positive in the code.

The average branch traversals for binary search trees also stay static with the increasing size of hash tables. This is because hash tables simply hold the trees, there is no association between them and the data they hold.

Discussion:

The analysis matches with the theory behind these ADTs. Increasing the size of the bloom filter would cause the number of lookups in the hash table to decrease because you would not have to check for as many false positives. Bloom filter size leaves the height of the binary search tree alone, while hash table size modifies it. The average branch traversals stay static with hash tables. One conclusion that can be drawn from this is the understanding that the hash table and the bloom filter are two independent abstract data types that simply work together to create this functionality of cryptography. This simple conclusion can be applied to all of computer science - individual parts and seemingly purposeless constructs can be created and combined to achieve whatever functionality the creator desires, a fitting paradigm for computer scientists.