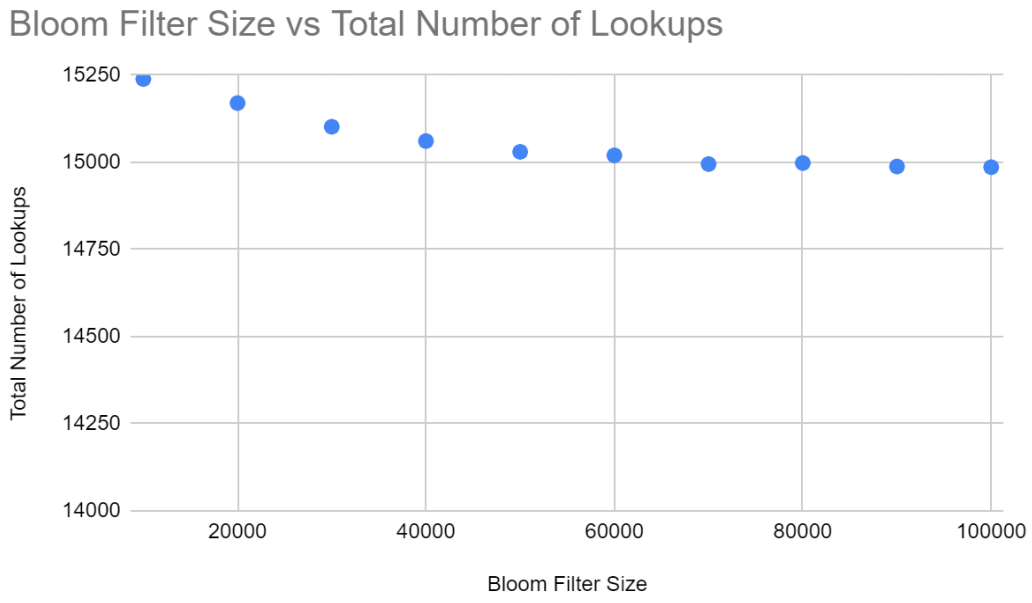
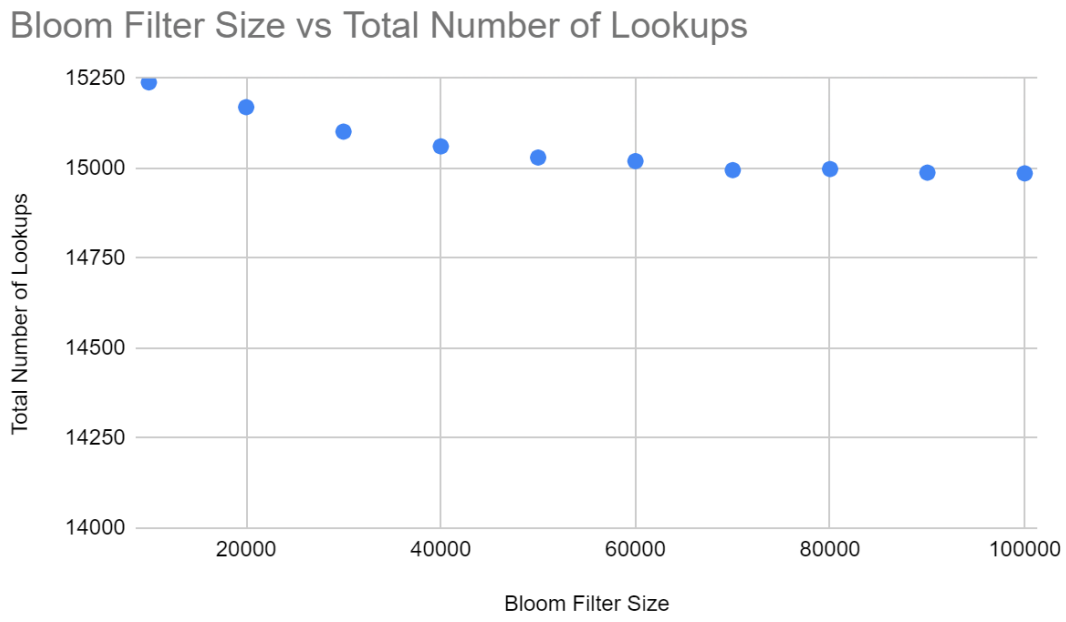
ASGN 7 WRITEUP

This writeup examines the efficiency of using different sizes of bloom filters and hash tables for a program that finds words in texts. I measured the total number of lookups, average number of branches traversed, and average height of the binary search trees at different hash table and bloom filter sizes.

How changing the size of the bloom filter and hash table affect the number of lookups?

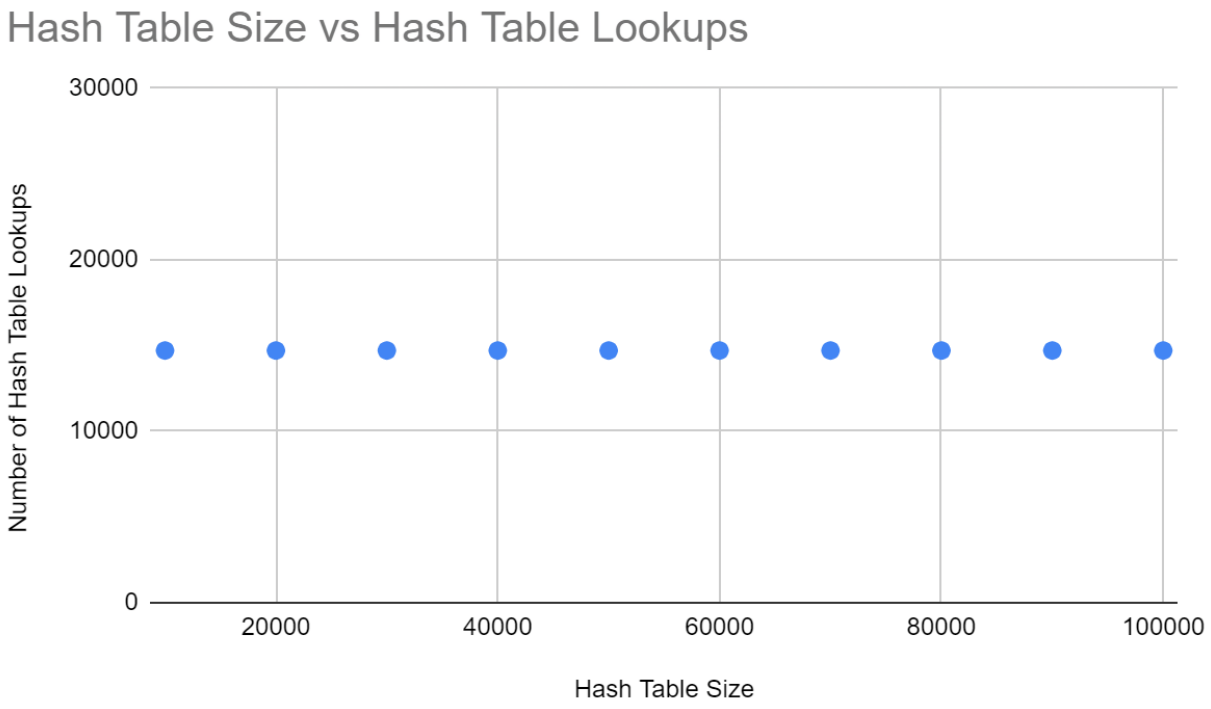
First I wanted to see how changing the size of the bloom filter and hash table would affect the number of lookups performed by the algorithm. While keeping the other data structure at its default value (2^{16} for the bloom filter and 2^{20} for the hash table) I changed the size while recording the number of lookups.





The first graph demonstrates that as the size of the bloom filter goes up the number of lookups in the hash table goes down. This makes sense because in bf_probe, bv_get_bit will be more likely to return a 1 in smaller bloom filter sizes which means there will be more false positives.

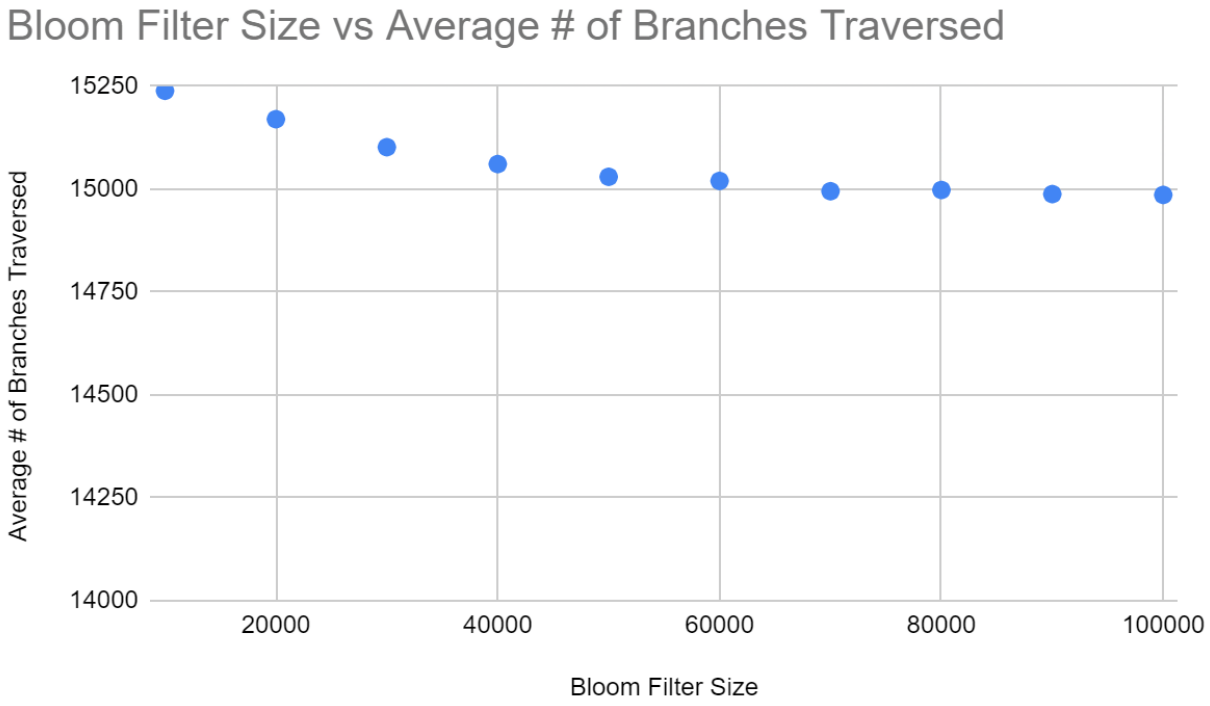
```
80 bool bf_probe(BloomFilter *bf, char *oldspeak) {
81
82     // hash oldspeak with each salt to calculate each indece
83     uint32_t i1 = hash(bf->primary, oldspeak) % bf_size(bf);
84     uint32_t i2 = hash(bf->secondary, oldspeak) % bf_size(bf);
85     uint32_t i3 = hash(bf->tertiary, oldspeak) % bf_size(bf);
86
87     // store the get bit result in uint32_ts
88     uint32_t probe1 = bv_get_bit(bf->filter, i1);
89     uint32_t probe2 = bv_get_bit(bf->filter, i2);
90     uint32_t probe3 = bv_get_bit(bf->filter, i3);
91
92     // return true if all of the bits were 1
93     if (probe1 && probe2 && probe3) {
94         return true;
95     }
96     // otherwise return false
97     return false;
98 }
```

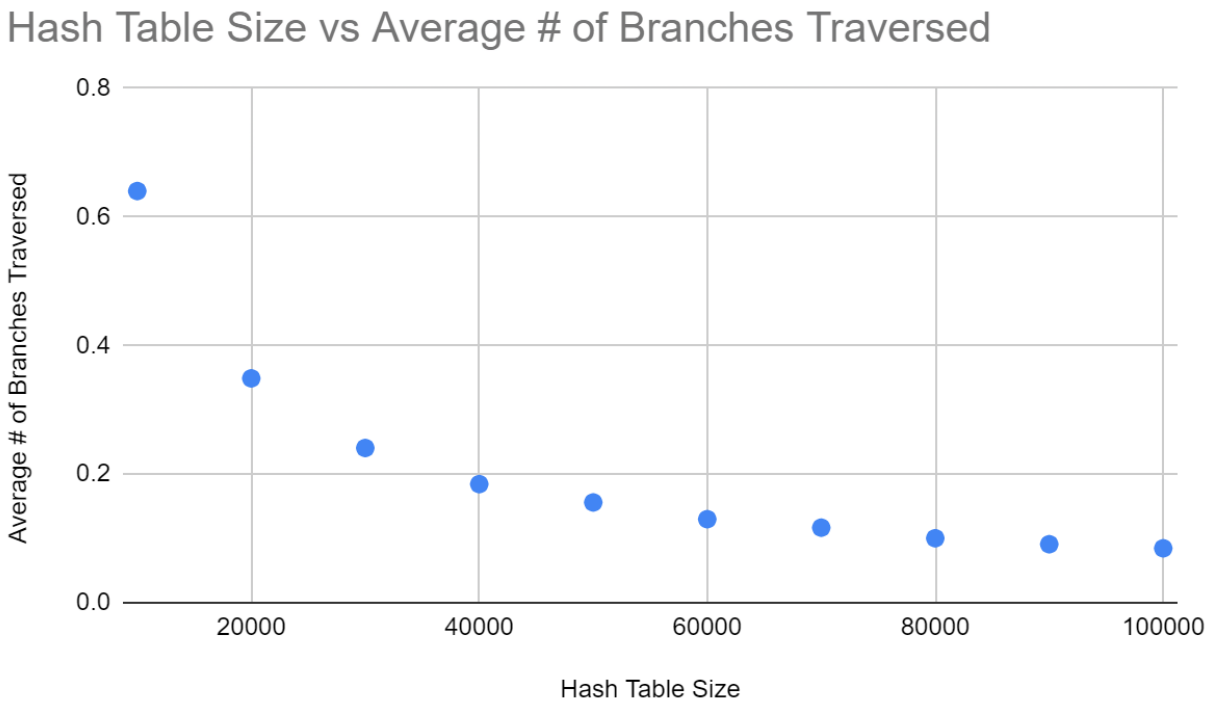


However, as shown in the graph above, when we change the size of the hash table, the number of lookups does not change. This is unusual because we would expect there to be a higher number of collisions with smaller hash table size and these collisions would cause us to perform more lookups. There may be some flaw with the way that I am incrementing the number of lookups that is causing this.

Average Number of Branches Traversed:
Another important measurement is the number of branch traversals that occur at different bloom filter and hash table sizes. In bst_insert and bst_find we use recursion to traverse binary search trees. These recursive calls can be computationally expensive if we are performing a lot of them, so understanding how the size of our bloom filter and hashtable affect the number of branches traversed can help us to make the program efficient.

```c
Node *bst_insert(Node *root, char *oldspeak, char *newspeak) {
    if (root == NULL) { // If the current node doesnt exist, insert the node here
        return node_create(oldspeak, newspeak);
    }
    if (bst_find(root, oldspeak) == NULL) { // if the node is a duplicate return root
        if (root) { // if the root exists, begin traversing the bst
            // use strcmp to go in the right direction
            if (strcmp(root->oldspeak, oldspeak) > 0) {
                root->left = bst_insert(root->left, oldspeak, newspeak);
            } else if (strcmp(root->oldspeak, oldspeak) < 0) {
                root->right = bst_insert(root->right, oldspeak, newspeak);
            }
            return root;
        }
        return node_create(oldspeak, newspeak);
    }
    return root;
}
```

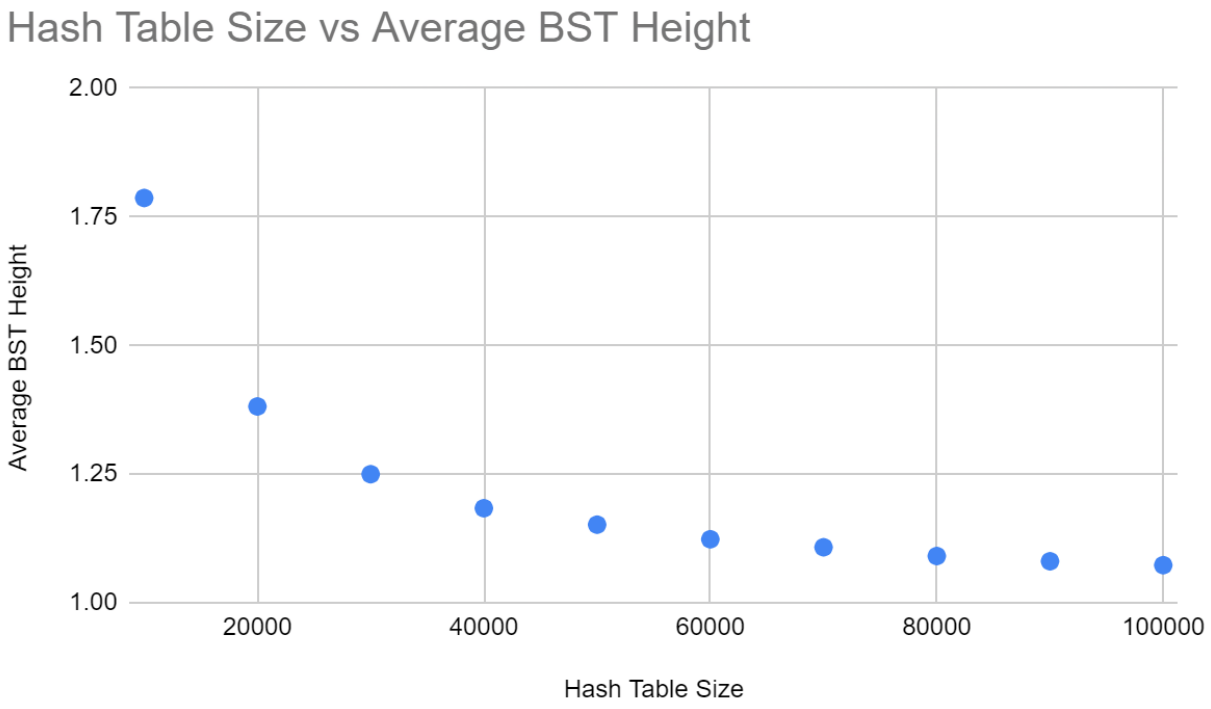


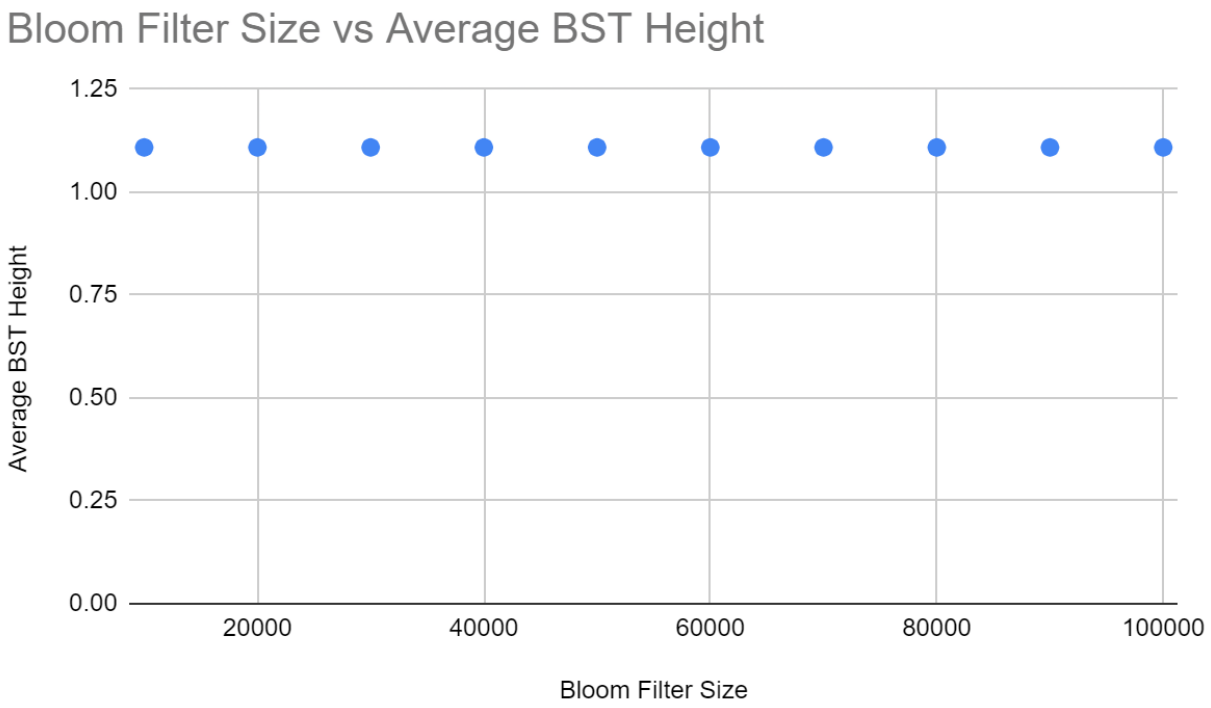


The first graph demonstrates how larger bloom filters reduce the number of branches that must be traversed. Larger bloom filters means that the binary search trees will not need to be as large so recursively traversing through them to either find a node, or to insert a node will not require as many recursive calls.

Similarly, in the second graph, increasing the hash table size also reduces the number of branches that need to be traversed. Increasing the hash table size means that there is more space for root nodes and the nodes can be more evenly distributed between many trees rather than a few large trees. Since we are traversing smaller trees it takes fewer recursive calls to find or insert nodes in the tree.

Average BST Height:
Lastly, I wanted to measure the way that bloom filter and hash table size affected the height of the binary search trees. When BST height is low searching for and inserting nodes is very efficient because we can simply hash the value we are looking for to find the right tree and then traverse the short tree to find the node. However, when the hash table size is small it means that the nodes will be concentrated into a few binary search trees which will be more computationally expensive to traverse.





As we can see in the above graphs, changing the bloom filter size has no effect on the height of the binary search trees, but changing the size of the hash table does. When the hash table is small, the number of collisions increases and this means that the binary search trees need to grow

to account for the extra nodes. This results in fewer, but longer binary search trees within the hash table that are more expensive to traverse than many shorter binary search trees.