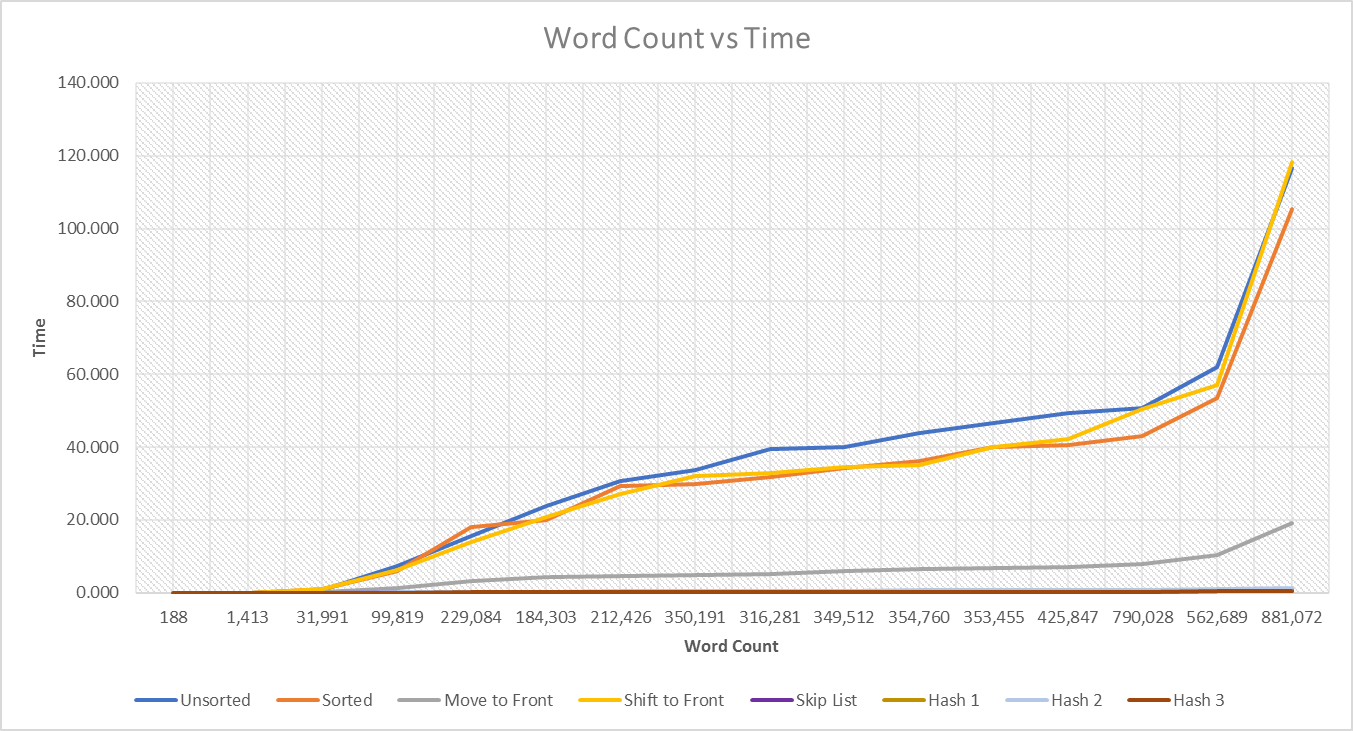
**Introduction:**

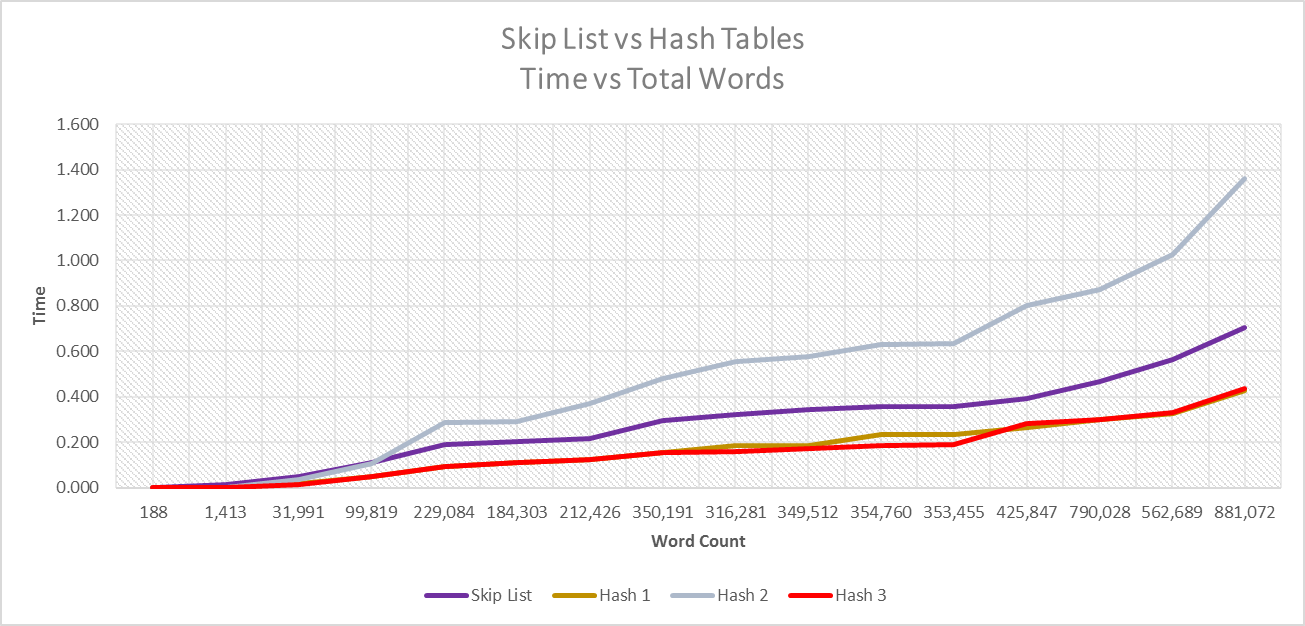
The objective in this lab is to observe the function and statistics of how the hash table works. The way hash tables work is that they receive an element of any type and take that element and manipulate its key into a location on the hash table, and hash that element into that location. We are to observe how well the hash table processes its data compared to the other types of list that we have collected data for in the previous labs. We are going to implement three different types of hash functions, which all do the same thing but the way the hash the element is different. The first method that we are implementing is where the hash will be the total ASCII value of the whole word added up mod 256 and it will be hashed to that location. The second method is only looking at the ASCII value of the first letter mod 256 of the word and hashing it to that specific location. Finally, the third method is code given by Dr.Thomas, where instead of hashing the word as a 32-bit value, we will only hash it as an 8-bit value. We will run the three different types of hashing on various text files and compare and dissect the stats that they are collected to the other types of list we had previously collected.

**Predictions:**

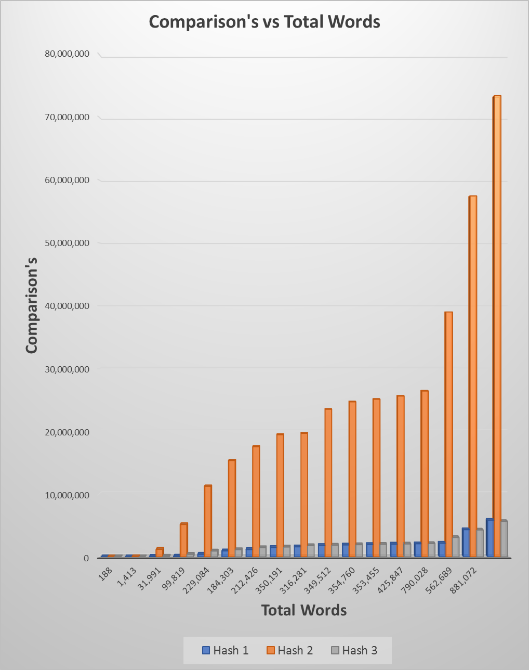
My predictions for this lab report, I feel are very accurate compared to the past lab reports since the logic for the hash functions make sense. I believe that method 1 with adding all the ASCII values in the word and modding it by 256 and method 3 with changing it to an 8-bit hashing will be the fastest. I think method 2 is by far the worse way to do it because you are only looking at the first letter of the word and modding it by 256 to that location on the table. Well if we think about it there are only 26 letters in the alphabet, therefore there are only 26 locations that any of the words can go to in the whole hash table! That’s awful if you are trying to find words quickly. That means that only 10% of the hash table is only ever going to be used at any time since it can only hash to 26 out of the 256 possible locations in the table. The first method breaks up the words better since there are so many different possible words that there can be, there is a slim chance that they are going to all be in one part of the table. As well for method 3, it takes into account the sequence of the characters in the word, which is AWESOME! In that cases that means the word “bat” and “tab” will hash to different locations in the hash table since it is sequence sensitive, which will allow for quicker look ups for a word and less comparisons when finding the word.

**Results:**

First I decided to take a look on how the different hash table methods compared to the other list based on time vs total words. As expected we can see that it blows the other list out of the water when it comes to putting the elements into the list. The hash tables are so much more superior when it comes to inserting the words into the list and how they are sorting them by having an array of linked list. The graph of the different sorting list are shown below:

Since it is hard to see what the hash tables and even the skip list are doing, I decided to put them in a separate graph so it is scaled better and get a clearly representation of how the list are really behaving as the size of the text files grow. As shown the hash tables are very fast even compared to the skip list and can even see that hashing method 1 and 3 are very fast compared to the skip list and barley go past 0.400 seconds when it gets to even the larger files which is crazy to think about since that is almost as fast as just passing through the data. The graph is show below of the skip list compared to the hash methods. BUT wait hashing method 2 is slower than the skip list! WHAT?!

Observing more in depth what hash table method 2 is doing, we can see why it would makes sense that it would be the slowest out of the rest of the 4 list compared to it. So thinking about it, if we are hashing the element to a location based only on the first letter of that word then as mentioned in the predictions there are only 26 locations that all the words will be hashed too. A demonstration of what it would look like is shown below:



|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |

“b”

“d”

“e”

“c”

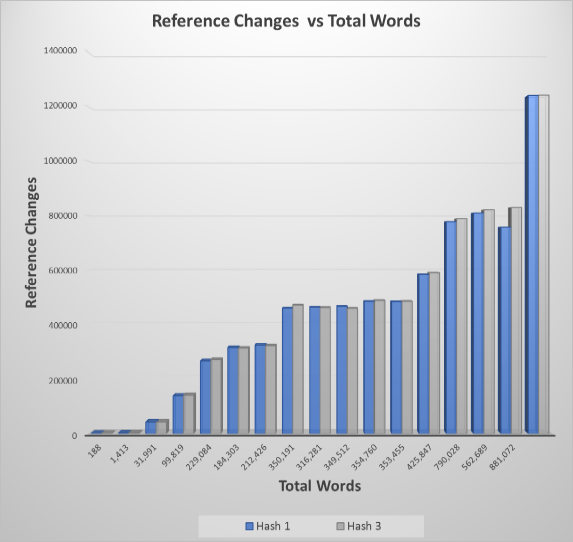
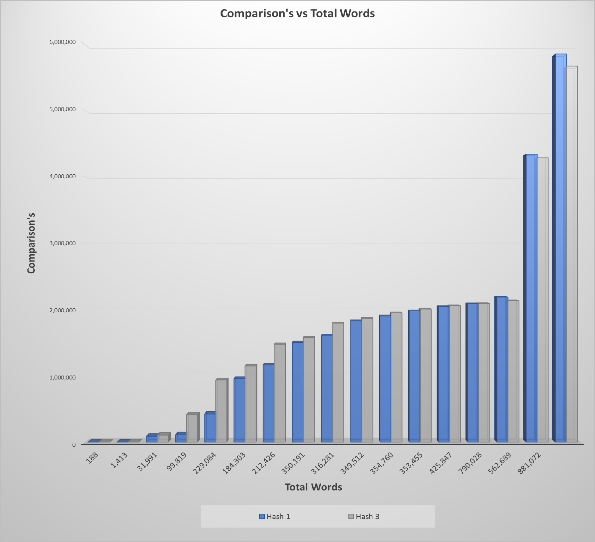
ddddddddddd

dd

Now, say that we have 8 possible cell locations that the words can be hashed too and that are vocabulary is only has 4 letters “b c d e” in it for simplicity. Our list will at most ever only use 4 of the 8 cells, meaning that it is only filling up 50% of the table at any time and all the words in the file would be jammed to four spots causing it to have a longer look up time versus that other hash functions that have a more spread out approach to hashing the elements in the table.

Also, we can see why the hashing method 2 would be the slowest out of all of them since it has so many comparisons compared to the other hash tables. We can see as the size of the file gets bigger the comparisons increase significantly which makes sense since all the words are in one part of the table instead of spread out like the other list are. As mentioned in the previous lab we saw a trend in the comparisons and the list itself, where they are directly proportional to the time it takes for the table to find the word it is searching for, since in the worst case scenario the word we are looking for could be all the way at the end of the list and have to make a lot of comparisons in order to make sure that the word is not in the list and would need to add it.

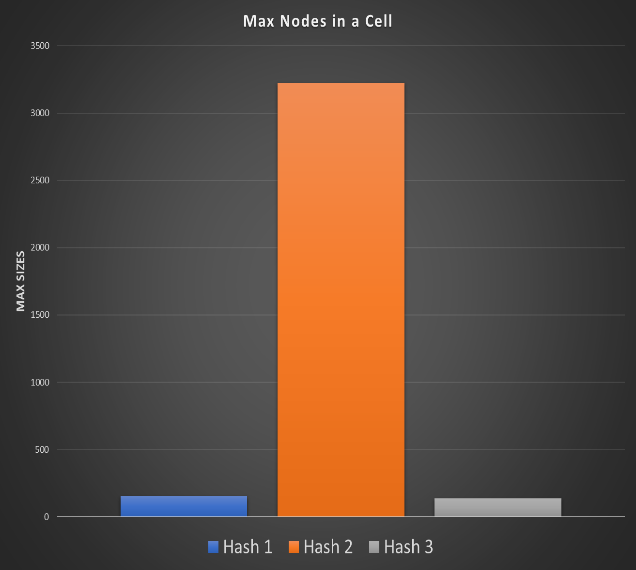
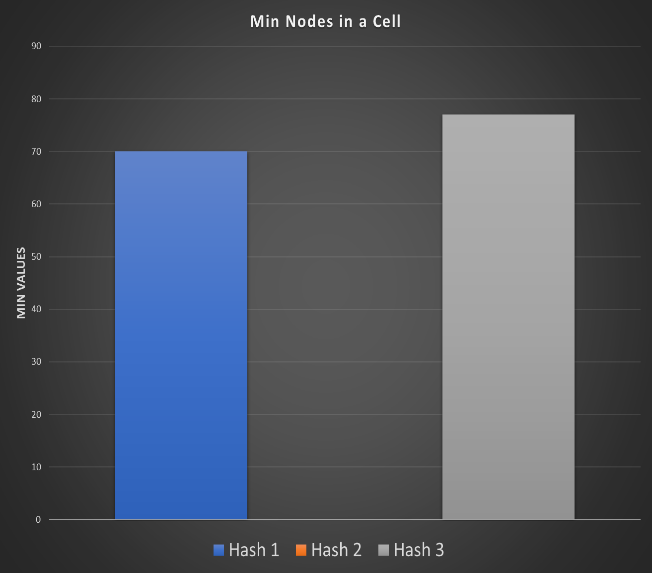
Next, looking at how the two fastest list compare to each other. They are pretty much neck and neck when it comes to processing the data and when it comes to time and how many comparisons it is making at each file. The code given to us by Dr.Thomas was as expected a little bit faster than summing the ASCII values of the word which makes sense since they are spread out more in the table and there is not as much clustering going on in the table. Shown below is the bar graphs of how the comparisons and reference changes for method 1 and 3 compare to each other. As said, they are pretty similar in both cases of comparisons and reference changes but the hash table 3 method starts to have better performance when the size of the file gets bigger.

****

When it comes to which out of these 2 are the best to use. Well I believe in these two cases it doesn’t matter since they are very similar, not in how they process the data but when it comes to process time and how many comparisons the table is making at a given time. Now if the file is really big and you want something that you know will process the data in a fair amount of time I would go with method 3 since as the files get bigger, it is still able to process the data consistently fast which is very nice.

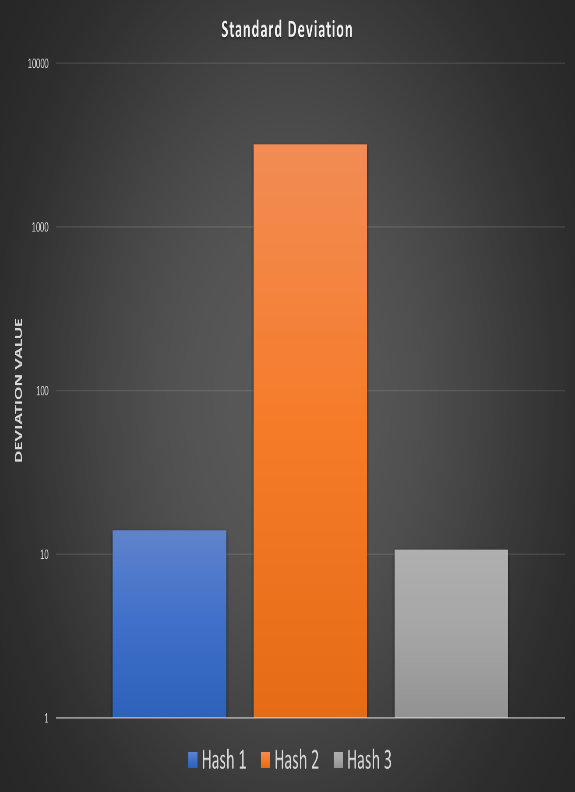
As well observing the way hash tables work, it makes sense that the less distinct words there are in a file and if the total words is decently low for that large amount of distinct words then the list will process much faster. Looking at the King James Bible we can see that the file would process fairly quickly since it has a total of 790,028 words and only 12,798 distinct words. That is crazy! That means if the words were split up perfectly evenly into each distinct word then that means each node would have about 61 words in them, which then all it would have to do is find the word in the short list and increment the count and it is done and goes onto the next word.

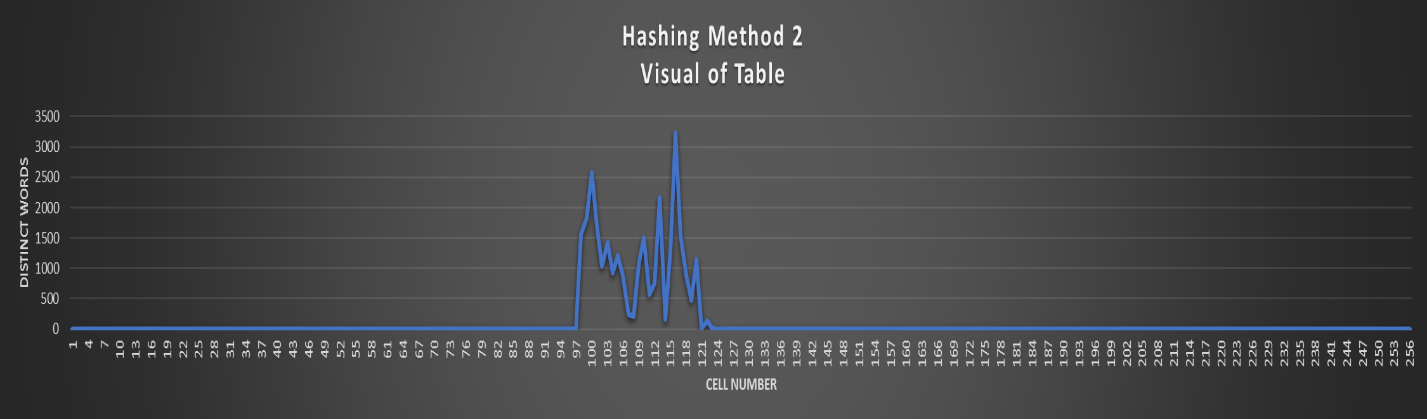
**Next we were asked to look at how evenly the list were split up across the hash table. Therefore I decided to test this theory on the Shakespeare file. First, knowing how the hash functions behave I decided to make a graph comparing the min and max values of the nodes out of all the cells. Before even knowing what the graph looks like, it makes sense that the second hash function would have a very high max value and very low min value. Shown below is the graph of the three hash table functions:**

****

**As shown by the graph it expected that method 2 would have the highest max value and lowest min value since the words are very distributed throughout the table. Method number 2 does not seem to be the best way since that its max cell has a total of 3,226 nodes in it and its min is zero should tell you that something is bad about this method and the words are not being spread out compared to the other two hashing functions.**

**We can see that the other two hashing functions are very nicely separated when it comes to the elements being way more spread out through the list. It makes sense that the closer your min and max are to each other the better you are at finding an ideal hash function that splits up the data that is being handed to it very nicely and would allow for easy and fast look up times. We see that the max value for hashing method 1 and hashing method 3 are 152 and 137, which is way better than 3,226. Then we can look at our min values and see that the mins are 70 and 77 for those list. Which in our case are relatively close in value and would inform us that the data is very evenly spread out, we can calculate how well it is spread out by calculating something called standard deviation which will tell us how well the nodes are spread out through the table.**

** Now, we are interested in the standard deviation of the words letting us know how well the words are really spread out in the table. Since we have a great deal of knowledge of excel thanks to Dr.Thomas for giving us practice, they have a very useful standard deviation function that we can use and give it a set of numbers to calculate for. So how standard deviation works is that it spits out a value that tells you how far away from the average the given values are. Meaning that the higher the number is then the more clustered and compacted the numbers are to each other and that they are not spread out well throughout the table. If the value that it spits out is low then that means the values in the nodes are somewhat close together and are spread out relatively evenly throughout the table and not consisting of clustering. As shown to the right we can see the standard deviation values that the three hashing function methods had generated.**

** Once again the method 2 has the highest standard deviation value. Since all the list nodes are clustered up in one part of the table and that makes sense since it can only go to 26 possible locations out of the total 256 possible cells that the word is able to hash to. Method 1 and 3 are about close in standard deviation, but as shown method 3 has a lower standard deviation value than method 1 and that just tells us that the words are spread out better in method 2. Shown below is the table of method 2 put into graph form showing where all the words are clustered at, and as it is shown we can see that they are clustered in the middle of the list and the rest of the 230 cells are all zero and not being used.**

**Conclusion:**

**Finally, the best part of the report, the conclusion where all the information is put together and predictions are proven. As shown by the data we can conclude that method 2 hashing table is not the best way to go if you are looking for your data to be spread out into multiple cells and have a low hash loading percentage. We saw by comparing the comparisons being made by the tables in the list and the method 2 was significantly high since it has to focus on one area of the truth table and has to search through a lot more nodes than it should have to and therefore making too many comparisons to be able to keep up with the other hash methods. We saw how the method 1 hashing and method 3 hashing where always neck to neck but as we got to bigger files, hash method 3 started to pull away since it splits up the nodes much better and able to access them much quicker since it is keeping the link list size relatively low.**

**We also saw that the hash table functions blew the other list out of the water when it came to time and how fast it was able to process the data. It is crazy how just the little changes in how a function works and changing its implementation for different data can make such a big difference. Now when we are talking about smaller files all the list were relatively close which makes sense since there is not that much data that has to be stored in the list in the first place. Even though the hash method 2 ran slower than the skip list, it still processed the files very quickly, just as expected as the size of the problem gets bigger the more time it will take to process that data. Now if you want process large amounts of data then yes it would make sense to look at hashing, specifically methods 1 and 3 since it processes the data very quickly even as the file itself becomes larger.**

**Looking back at the predictions at the beginning of the report, they were pretty head on in terms of the behavior of the hash tables and which one would be the best and which one would have the slowest time. We proved that only looking at the first letter of the word and parsing it to that location on the table based on its ASCII value would not produce so great of results. We proved that was the case by looking at the comparisons that were being made and also what the table actually looks like in terms of where all the words are being stored and that they created one giant cluster. Then, we were able to calculate the standard deviation of the data and see that based on the value given we can see how well our data/elements were spread out into our table, and see which method would be better to implement based on the lower value.**