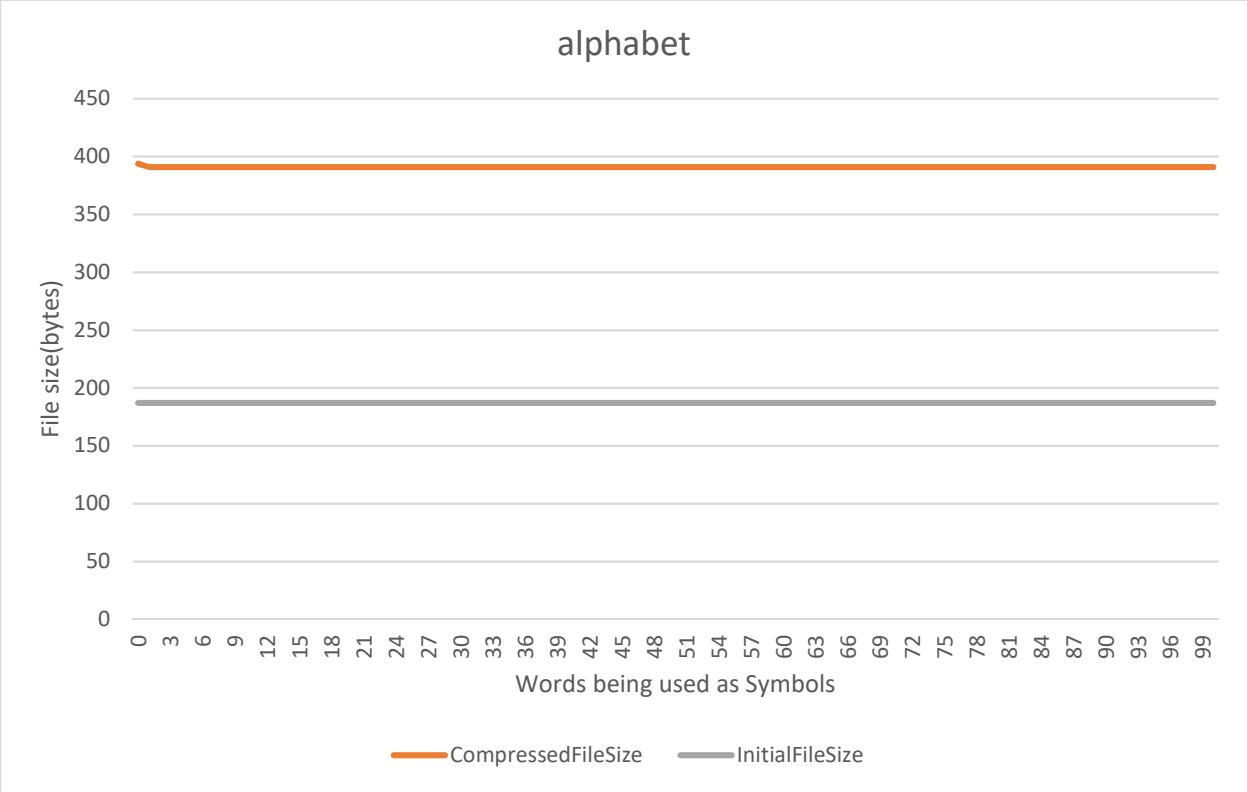
Nickolas Komarnitsky u0717854
CS2420 Assignment 12 Huffman
04/21/17

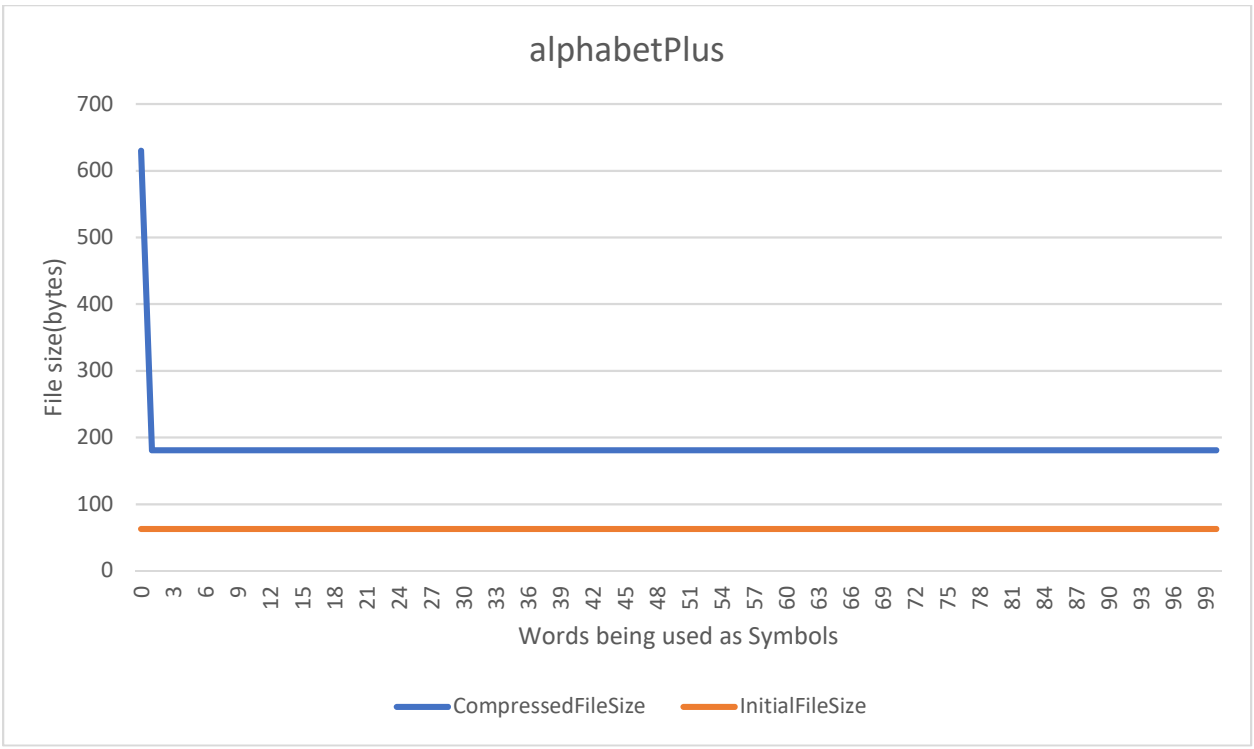
In this assignment I learned a lot more about the low-level functions of a computer. I learned how to properly use a bit set, and Byte classes. I learned how to easily construct a Huffman tree. It took a bit of work to get this assignment going, but overall I learned a lot. A Huffman tree is like a binary search tree, except data is only in the leaf nodes. The root node acts as a starter. To find something in the tree you use a bit code, a string of 1's and 0's. If there is a 0 you go left in the tree, a 1 you go right. This should lead to a leaf node with the value you are looking for.

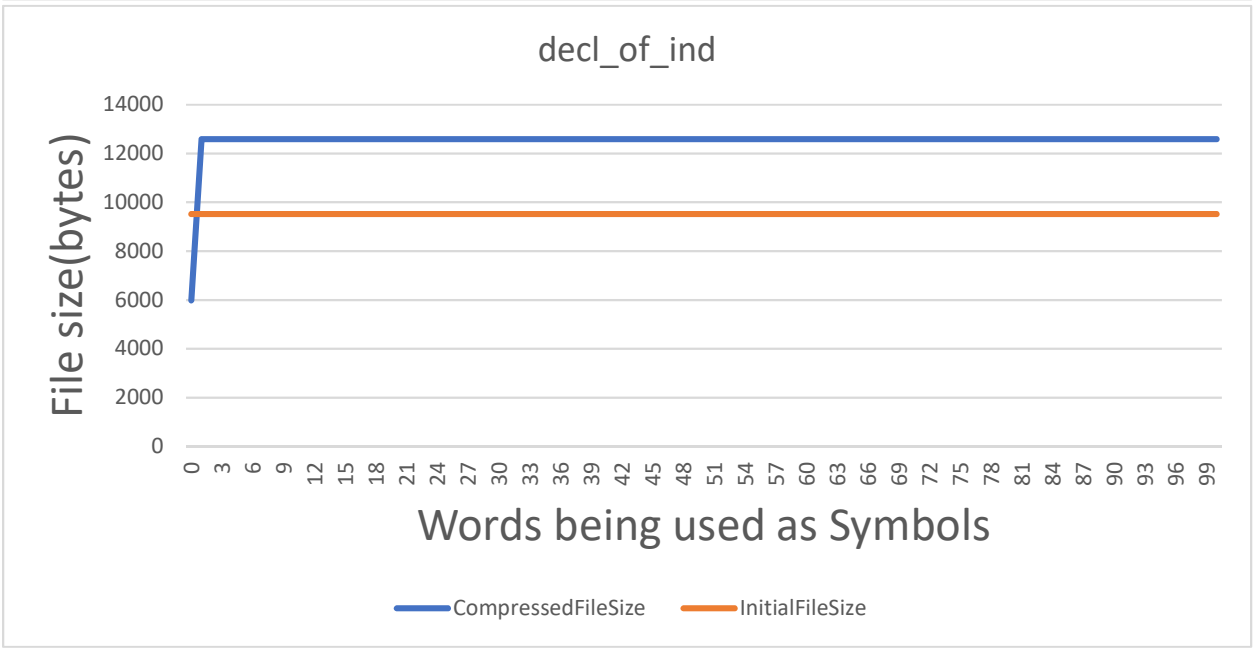
# Complexity Experiment

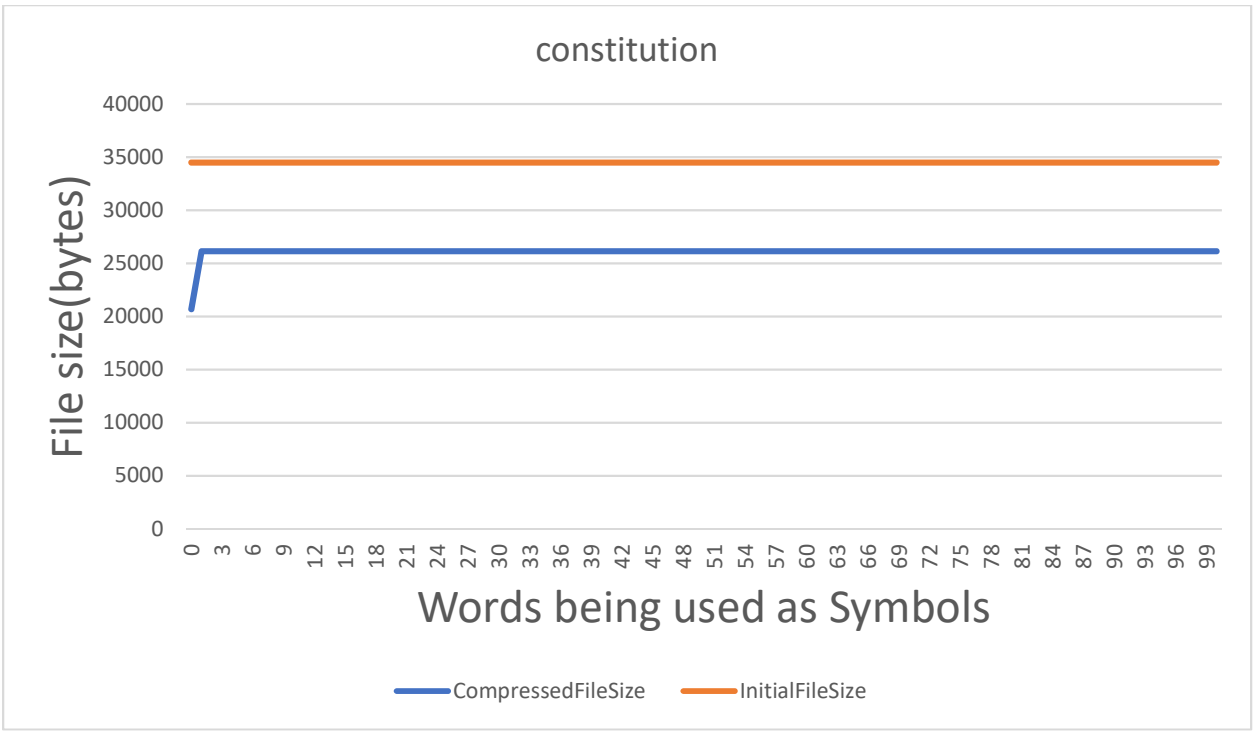
For my experiments I tried to find out if this was a good compression algorithm. The first experiment I did was to see if the file was getting compressed well. I tested word count along with this. In the actual experiment I iterated through 0 to 100 words being used as symbols. Then recorded the time it took, the initial file size, and the compressed file size. I did this for each of the files provided.

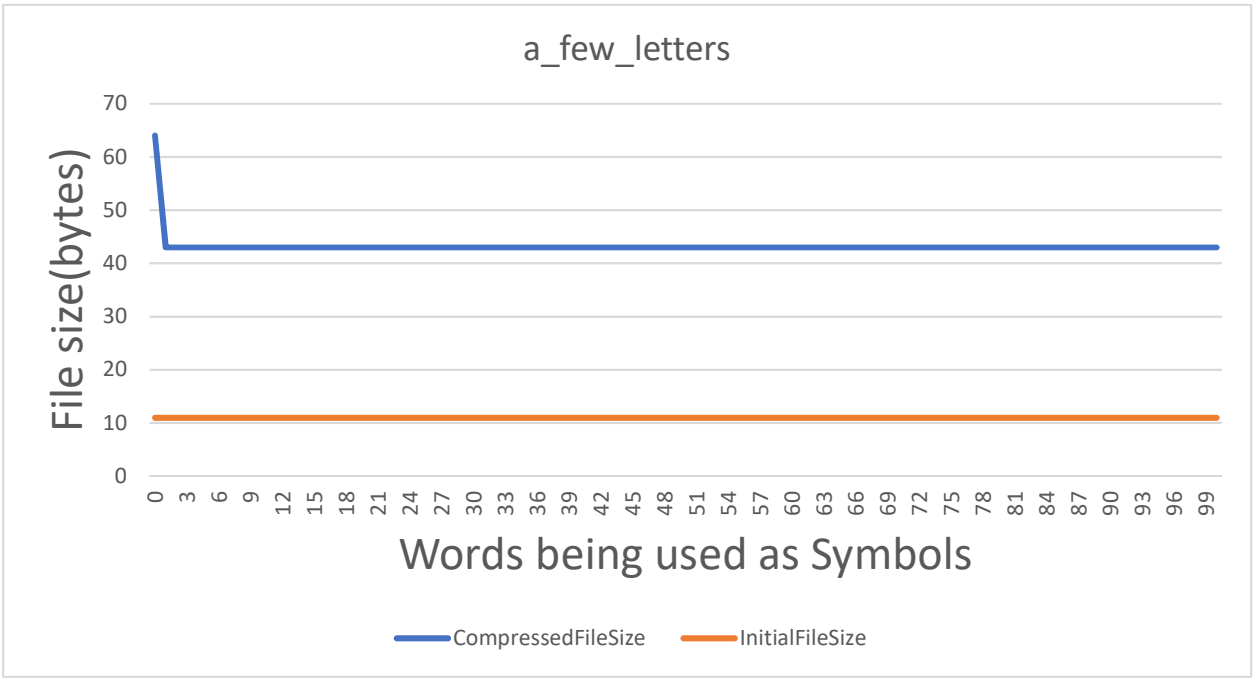
# Graphs-File Size

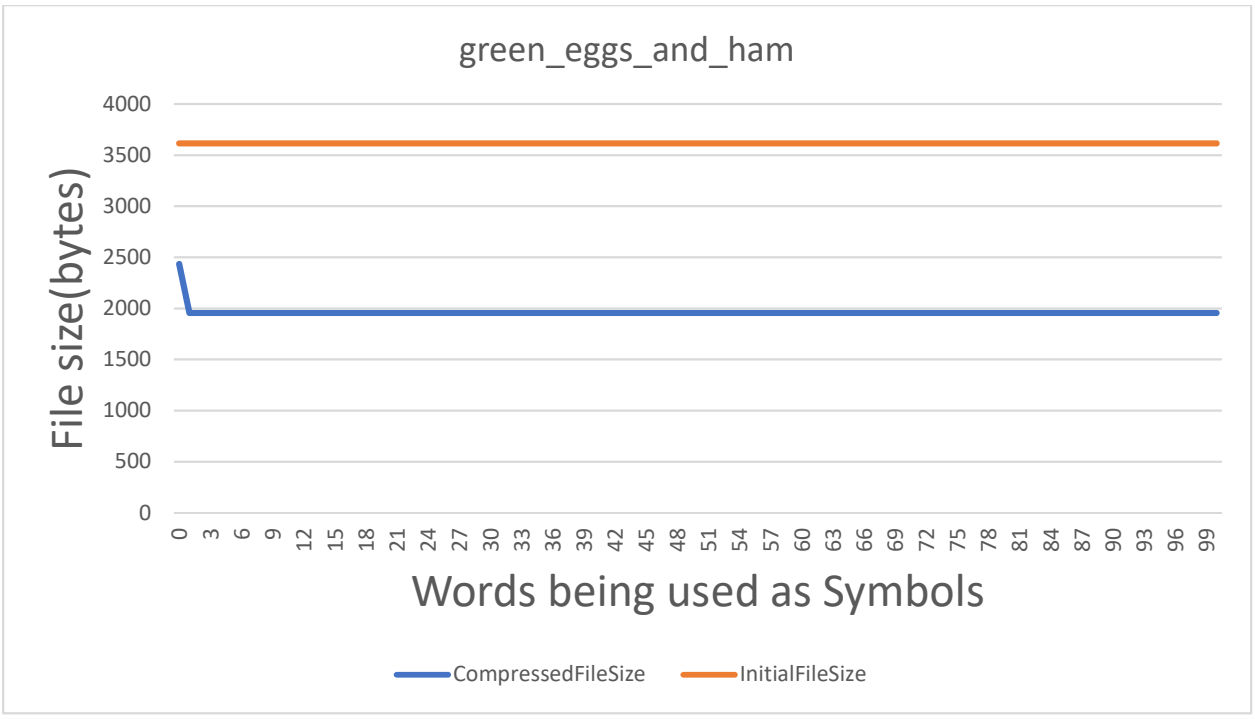


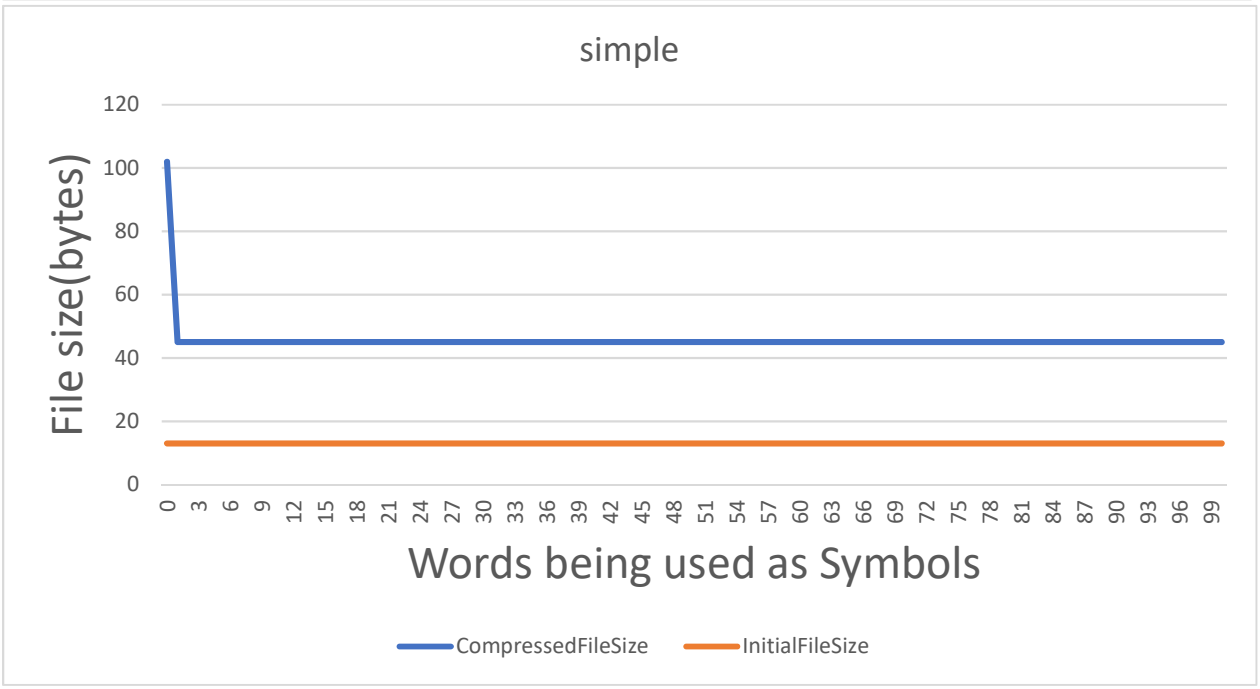

alphabetPlus
700
600
500
400
300
200
100
0
File size(bytes)
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

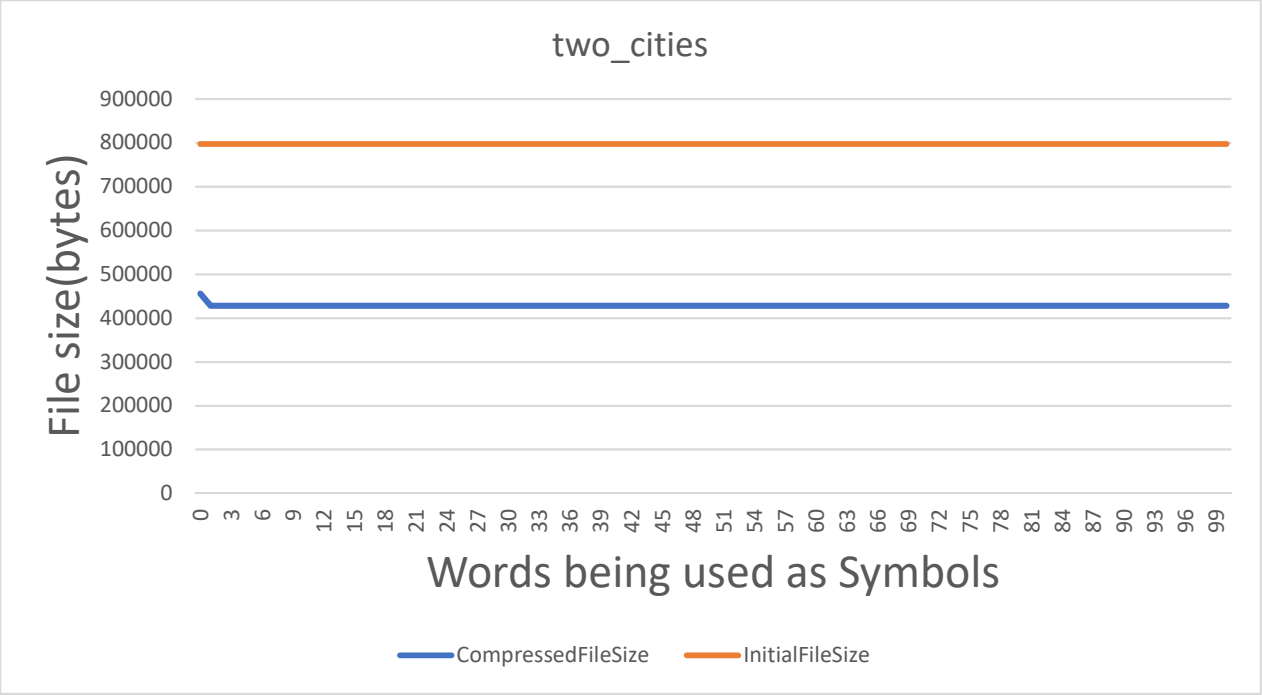

decl_of_ind
14000
12000
10000
8000
6000
4000
2000
0
File size(bytes)
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize


constitution
File size(bytes)
40000
35000
30000
25000
20000
15000
10000
5000
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

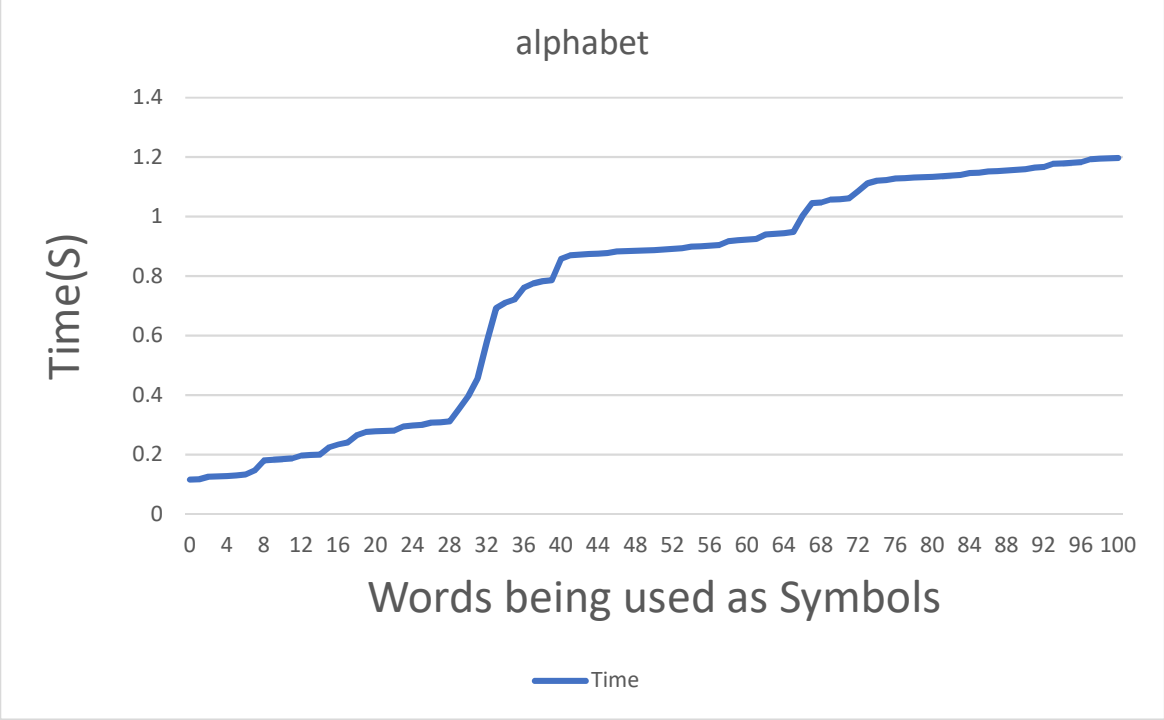

a_few_letters
File size(bytes)
70
60
50
40
30
20
10
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

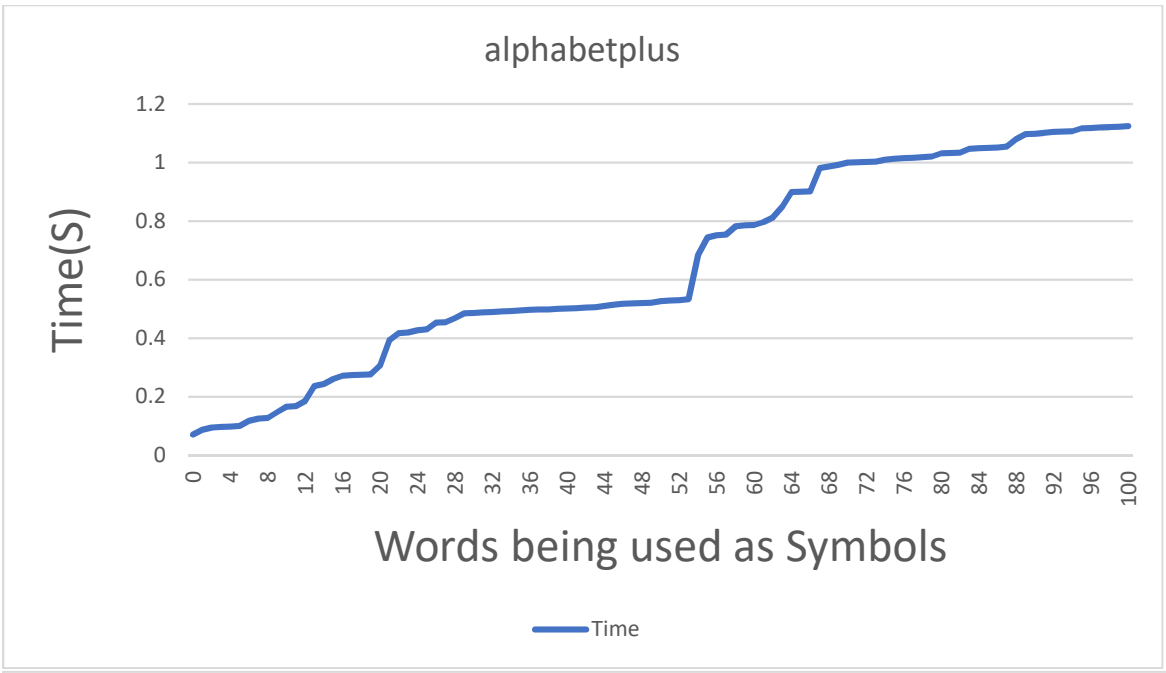

green_eggs_and_ham
File size(bytes)
4000
3500
3000
2500
2000
1500
1000
500
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

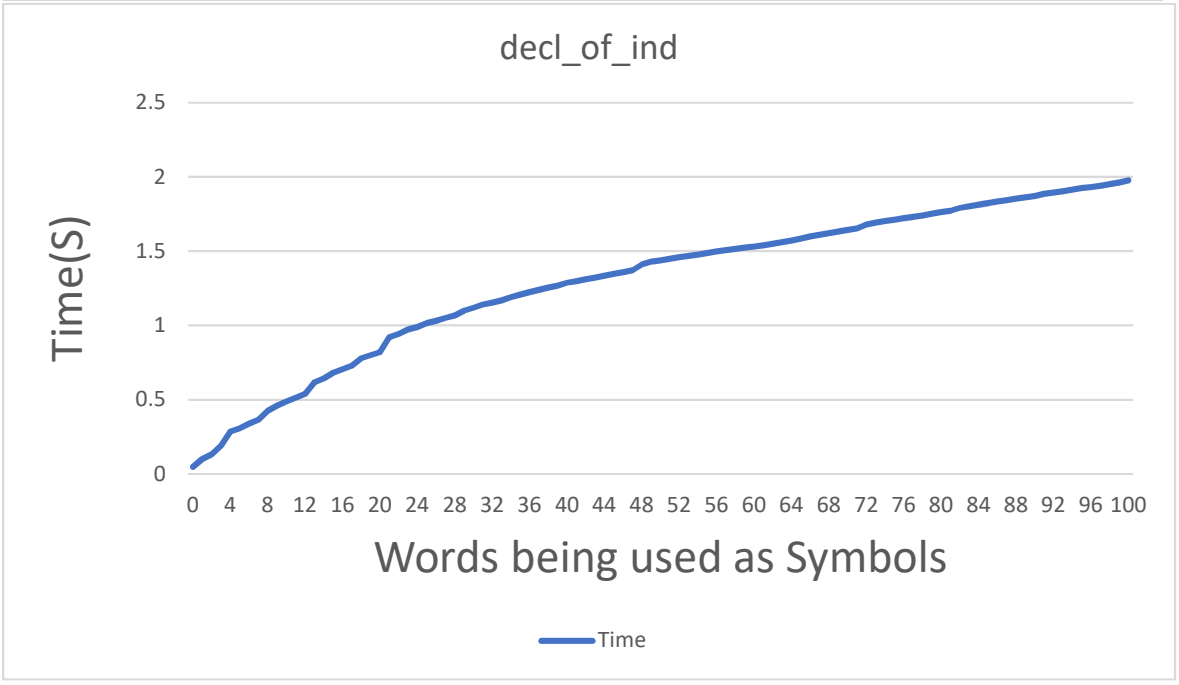

simple
File size(bytes)
120
100
80
60
40
20
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

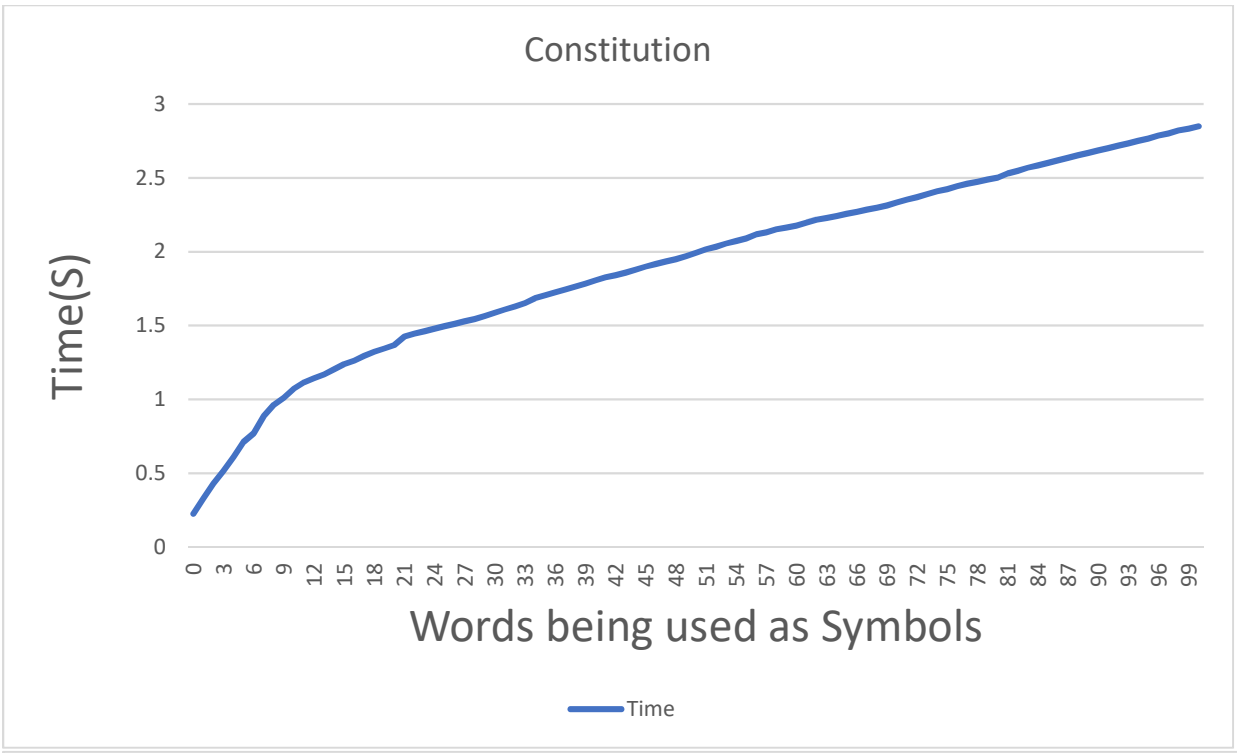

two_cities
File size(bytes)
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
CompressedFileSize
InitialFileSize

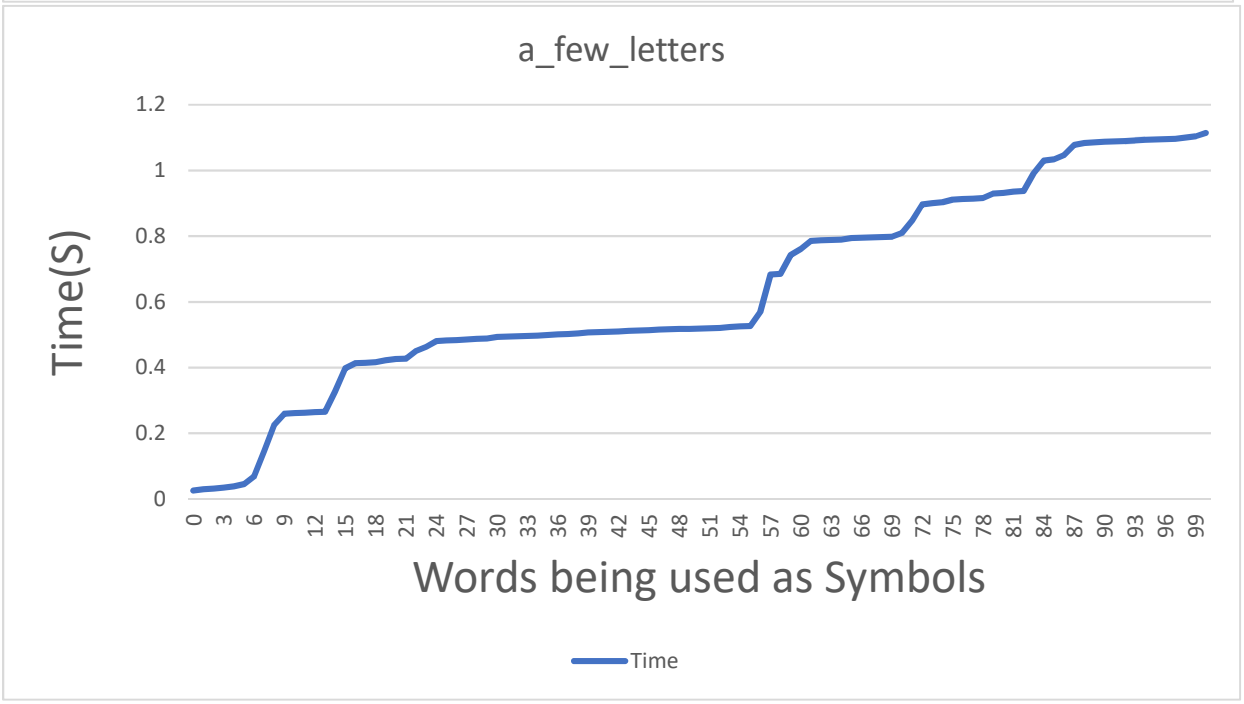
# Graphs-Time

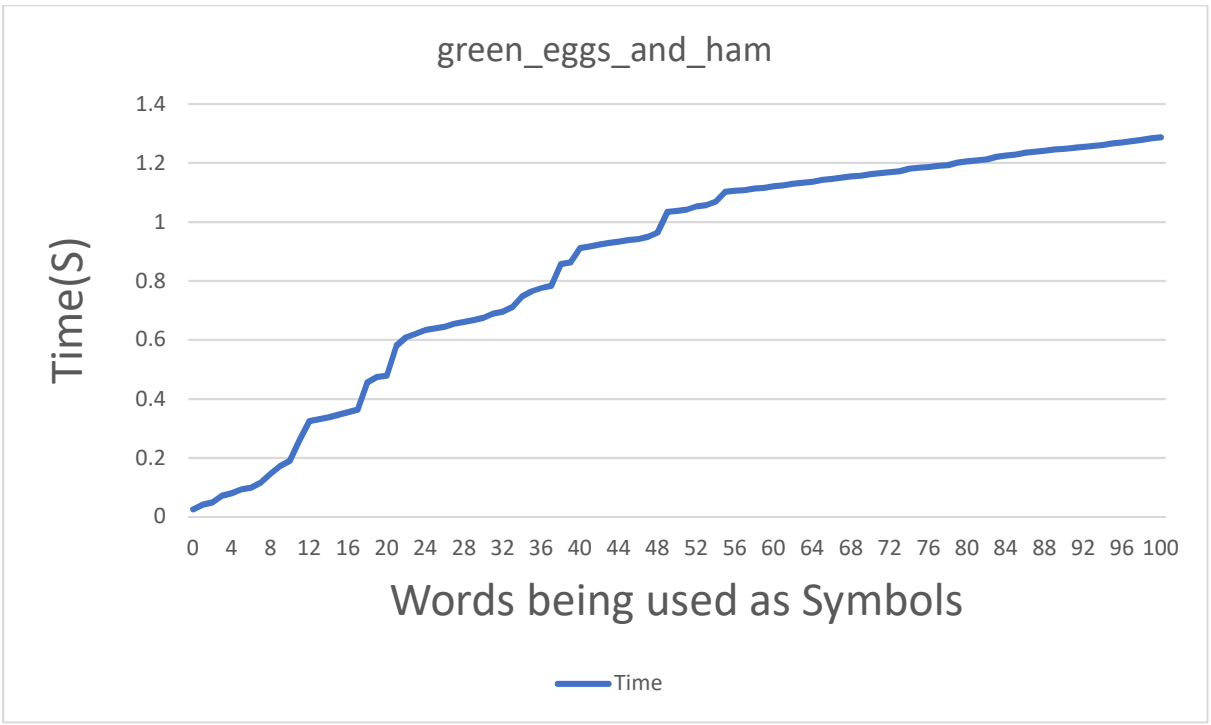

alphabet
Time(S)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100
Words being used as Symbols
Time

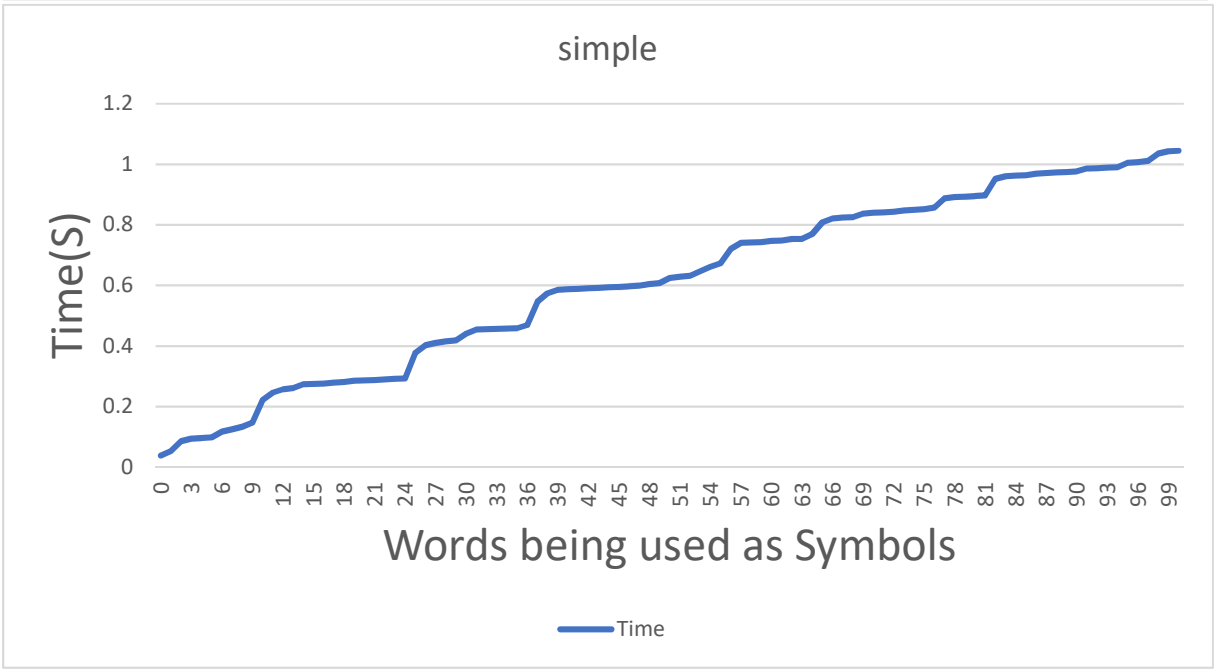

alphabetplus
Time(S)
1.2
1
0.8
0.6
0.4
0.2
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100
Words being used as Symbols
Time

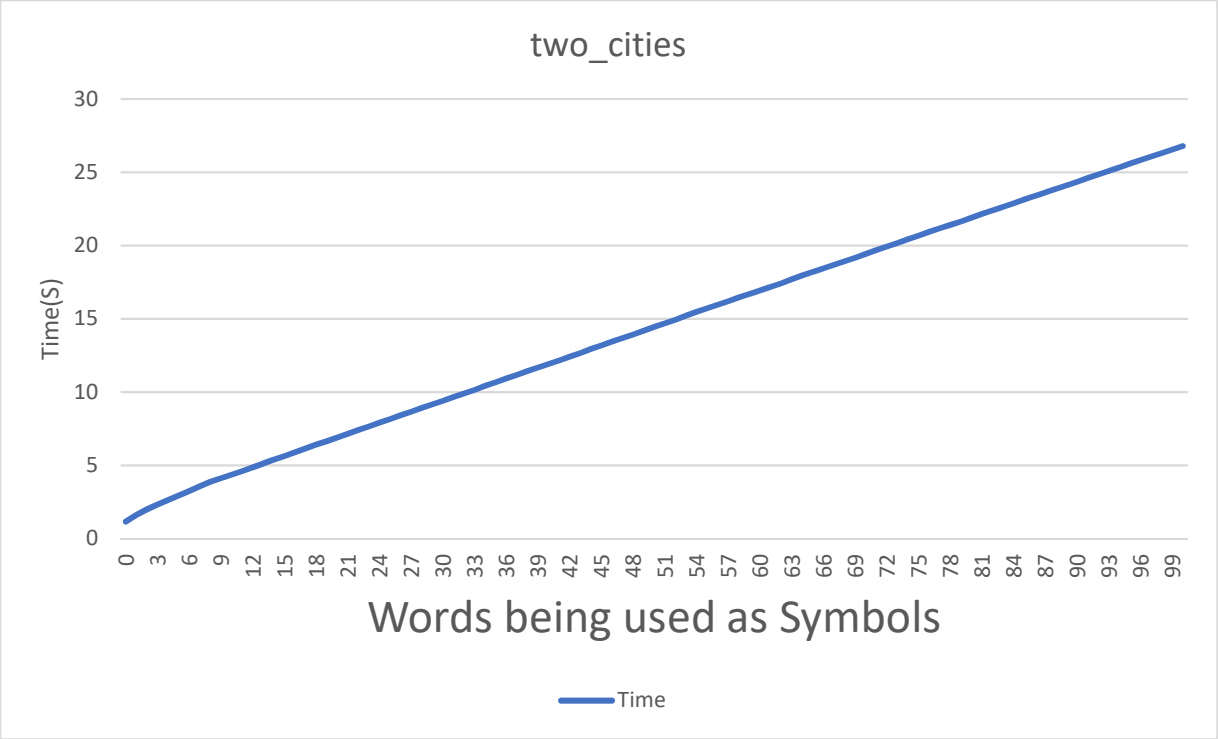

decl_of_ind
Time(S)
2.5
2
1.5
1
0.5
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100
Words being used as Symbols
Time


Constitution
Time(S)
3
2.5
2
1.5
1
0.5
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
Time


a_few_letters
Time(S)
1.2
1
0.8
0.6
0.4
0.2
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
Time


green_eggs_and_ham
Time(S)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100
Words being used as Symbols
Time


simple
Time(S)
1.2
1
0.8
0.6
0.4
0.2
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Words being used as Symbols
Time




# Graphs-Summary

The above graphs show that this algorithm depends heavily on the size of the file. When the algorithm is run on a file with one or two words/letters the compressed file ended up being much bigger that the original(a_few_letters, alphabet, alphabetplus, simple). The Declaration of independence file however was only smaller when using 0 words as symbols. For every file, the size when using 0 words was vastly different from the size when using 1-100 words as symbols. The very large files(constitution, two_cities, and green_eggs_and_ham) came out with smaller compressed files. The conclusion drawn from this is that this algorithm is better the larger the file. If the file is only a few words than it creates a bigger file and that isn't compression. The timing graphs all show a complexity of O(NlogN) with N being the number of words used as symbols. The more words being used as symbols the longer it takes to process the file.

# Software Engineering

This assignment took me 15 – 20 hours. It was a much harder assignment than the rest. I struggled getting the bits to write properly for the body of the text. The easiest part was getting the tree working though. I did attempt to make a gui for this, but it just wasn't coming together fast enough. I did trim down my timing tests even more for this assignment, streamlining them into a single parent class, and then a child class. This was simplest for the timing because I ran only one style of test, so all I had to do was make on class to do the testing instead of multiple like the previous assignment. I also update the parent class to have more functionality.

**Bitsets**

I learned a lot about bit sets in this assignment. I knew nothing about them except for the basic, it's a string of zeroes and ones. The functionality wasn't easy for me to get at first, but I got it down after studying the api. I learned how to get what you wanted out of the bitset.

**ByteBuffer**

I learned about how a byte buffer reads and processes bytes. The functionality of the buffer was fairly easy to get the hang of, but it didn't seem to be working correctly in my code. I had to work on it for a bit to figure out exactly how to fix it. The buffer didn't function the way that I originally assumed that it did.

**ByteArrayInputStream**

This was used to write bytes to a file. First time using it I learned how it stored everything and what was written into the array it contained. Learning how this functioned properly made it easy for me to get the writing correct for the header.