Lab 3 explanatory text (examples of executions in the bottom)

Lab 1

To solve this assignment I made a recursive function that lets the user input chars until he/she enters the enter-key (assignment didn't specify when to stop letting the user input so I thought this was a logical way to let the user stop writing). In every recursion of the function it is checked if the input is a letter, blank space or newline(can only be a newline the last time), if so we print out the char with "putchar" and if else we "putchar" a blank space as the assignments explains that you should. What the function does is to check if the input was a newline, if so we stop the recursion and if it is not a new line we simply call the function again (recursion happens).

Lab 2 (tables and graphs in Lab 4)

In this lab the main complexity was creating a binary search symbol table and using it together with the frequency counter. The binary search symbol table has three class variables: keys, values and n. Keys is an array with all the keys (keys are a Comparable type object), values is an array as well (with comparable objects as well) and lastly n is simply the amount of keys/values in the binary search symbol table. You can either create a binary search symbol table by not having any parameters. The most important methods in this class are: put, get, delete I would say, therefore I will explain them. What put does is simply to add a key and value to the symbol table (in its correct place since the keys are in sorted order), this is done by checking where they key parameter should be put using a rank method that checks what index it should be put to, after this we simply check if the key is already in the keys array or not and if not we add it and if needed we size the array up so it fits. The get method is used to return the value that the parameter key has bound to it, this is done by checking where the parameter key would be set in the array using the rank method and then if that index has the same key that key's value is returned, otherwise null is returned since the key doesn't exist. Lastly the last very important method is delete, which as it sounds like deletes the key that is sent in as a parameter and also its value associated with it. This is done by checking the parameter key's rank and then seeing if the key in that index is the same as the parameter key, if no we simply do nothing since there is no key to remove but if it is the same key then we remove that key by setting every index from that point in both keys and values to the index after it, and deleting the one we started with.

The second most important part of this assignment was the frequency counter so now I will explain how that class works. This class has the main method in it, which is from where I read "*the text*", filter the text so I get all the words in it, and then put the first 1000 words into a binary search symbol table, lastly the main method finds the word that was most common and also prints some stuff in the end that I will explain more in detail soon. The binary search symbol tree that is created is created with Strings as the keys and Integers as the values since the keys are going to be the words and the Integers are going to be the amount of times the word has been found in the first 1000 words of "*the text*". The file is read using a scanner that scans the next line until there aren't more lines to scan, and for each line scanned the line is examined by a for-loop that goes through every character in the line and checks if it is a letter (using the Character.isLetter(*char*) method), if it is then its added to a string and if not it is simply ignored. This string is then split by "(\\s+)" since this splits the string into a string array with one or multiple spaces as the separator between words that are put into each index of the array. Then a for-each loop goes through every word in the array and basically adds it to the created binary search symbol tree, if the key already existed 1 is added to the value and otherwise it starts at 1. This is done until 1000 words have been added to the binary search symbol tree. Worth noting is also that a stopwatch class was created before beginning to read the first 1000 words from the file and after being done with it the time is saved as a double. The next thing in the main method is that every word in the binary search symbol table is gone through by a for-each loop and if that word has the highest value connected to is recorded it is saved as the max value. This examination of finding the max value is stopwatched in the same way as reading and inserting the words was. Lastly the main method prints out the max value and the key (word) connected to it, the number of distinct words(added 1 to a counter every time a new distinct word got added), the number of words and the time it to to get the max value and also the time it took to read and insert all the words.

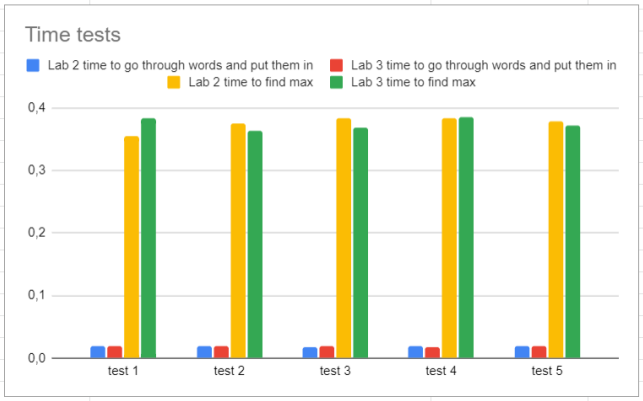
Lab 3 (tables and graphs in lab 4)

This Lab was very similar to the second one so I will just go more in detail about the differences instead. The assignment was the same besides that instead of using a binary search symbol tree in lab three a binary search tree is used. The main difference between those two is that the binary symbol table uses two arrays to store the keys and the values, while the binary search tree uses nodes. A node in this case is an element which contains a key, a value, a size, a left and a right. The key and value we have already heard about however the rest are new. The size is simply the number of nodes in the substree (so all nodes "under" a specific node), the left is the node "down to the left" (which is smaller in key value) and the right is the node "down to the right" (which is larger in key value). So basically it is called a tree since every node is linked to the left and the right somehow and therefore we get kind of a tree-like structure with one node at the top and further down there are a lot more since most nodes have a left and right

that go “down”. Besides this the two assignments are almost identical besides that of course the methods have to be redone a bit to suit the different way of storing the keys and values in the binary search tree.

Lab 4

Lab 2 time to go through words and put them in	Lab 3 time to go through words and put them in	Lab 2 time to find max	Lab 3 time to find max	
0,019	0,019	0,355	0,384	test 1
0,02	0,019	0,376	0,363	test 2
0,018	0,02	0,383	0,368	test 3
0,02	0,018	0,383	0,385	test 4
0,019	0,019	0,379	0,371	test 5
Average	Average	Average	Average	
0,0192	0,019	0,3752	0,3742	



Results:
https://docs.google.com/spreadsheets/d/1irvNEPAdeA1CZmI66F9Mrn3vWgSQItCm6ZlKnnpA744/edit?usp=sharing

In these results we can see that the difference is extremely small, to such a degree that I would consider them the same. So in other words at least when doing the measurements for 1000 words the time complexity is pretty much the same for lab 2 and lab 3. However now the important thing is understanding why this is, why the execution time doesn’t change even though we compare a binary search symbol table and a binary search tree.

The main difference in the first test (the time to go through words and put them in the symbol table) is the put method in lab 2 and 3. And if you look at the put method in both you can see that both have the same time complexity, which is O(N) in the worst case, there are minor differences of course but since both are closest to O(N) the difference will be very small, especially when O(N) is the worst case and we are handling so few words in this test. The get method is also used (in the contains only get is used) both lab 2 and 3’s get methods have the time complexity O(N). This is since in lab 2 the method calls itself at maximum N times (if it’s a totally unbalanced tree). And in lab 3 the time complexity should be log(N) since it's a binary search. But when summing it all up we

can see that both lab 2 and 3 have some kind of O(N) time complexity which is why they have a very similar result in the test.

The main difference in the second test (the time to find the max) is the keys() method implementation in lab 2 and lab 3. In lab 2 the time complexity is O(N) since we go through every element once and the rank method goes through them a bit less than N usually (max N) and N+N is still O(N). In lab 3 the time complexity is O(N) as well since we call the key method more than N times, but not close to N^2 or something similar. And since we have so few elements then we could say both O(N) complexities are the same which we can also see is true in the tests. However to find the max get is also used a lot, but this we have discussed in the previous paragraph so it doesn't have to be mentioned again. And summing it all up we can see again that both lab 2 and 3 have time complexity O(N) which is why they had very similar results in the test.

Lab 5

This question wasn't very clear so I had to do my own evaluation of how to measure how evenly the built-in hashcode function for string in Java distributes the hashcodes for the words found in the text. My idea was to have the same implementation as in Lab 2 but in addition to creating a binary search symbol table for the words in the main method I also created one for their hashcodes, this will use the hash codes as keys and the values will be how many times they appear. This way I can see how many times I get the same hashcode if I take the number of distinct hashcodes minus the number of distinct words (since obviously the same word will have the same hashcode). In addition to this I also checked how many hashcodes where in the span of every tenth of the full hashcode range I got, so if my min hashcode was -10 and my max hashcode was 10 then I would cut that range into 10 different parts and see how many hash codes were in all parts. This was done by finding the min and max hashcode (highest and lowest key) from the binary search symbol tree (similar to how I found the max value in Lab 2, but instead of the value I just checked the key). Then I found the range by taking the max hashcode minus the min hashcode, this number will be very large so I stored it in a double. Then I went through every hashcode key in the binary search symbol tree with hash codes and checked if they were "smaller or equal to" the min hash plus a tenth of the hash range times "i", where "i" was the counter in a for-loop, if so 1 was added to that index to a 10 size array that started with 10 zeros. Lastly the important things that are printed at the end of the main method are: the number of words, the number of distinct words, the number of distinct hash codes, the min hash, the max hash, the hash range and the number of hash codes in every tenth of the hash range.

As we can see in the results the hashcodes are pretty evenly distributed, besides the 6th tenth that has a lot more, and since it's a pretty big sample size this is a relevant test which does show something.

Lab 6

In this lab the main complexity was creating a separate chaining hash symbol table, sequential search symbol table and main method used together with them. How it works is that the hash symbol table could be seen as containing several rows where each index is a hash code and these rows are a sequential search symbol table. I will start with explaining the sequential search symbol table. This class works as a list with keys, and it uses nodes to store the values and keep the list in place. Every node has a key, a value and a next. The key and value are the same as the other labs and the next simply points to the next node in the list. Besides this the class has the usual put, get, delete as the main most important methods. The difference here from the other labs is that the keys are simply put to the first place in the list. Now I will explain the separate chaining hash symbol table. This class has three class variables: n, hashTableSize and symbolTable. The variable n is an int that represents the number of key-value pairs in all "rows/indexes", so all key-value pairs that exist. The int hashTableSize is simply the number of "rows/indexes", so the number of different hash codes that exists, and all of these hold one row which is a sequential search symbol table. The last class variable is symbolTable and this is simply a sequential search symbol table (one of the "rows/indexes"). Besides this the separate chaining hash symbol table had the usual put, get and delete as the most important methods. The main difference with these methods here compared to the normal binary search symbol table in lab 2 I would say is that before the key is put in its place it is hashed using a hashing method, and of course this means when you get the value from a key then you have to first hash the parameter key and then get the value of that cash code since in the list there are hash codes.

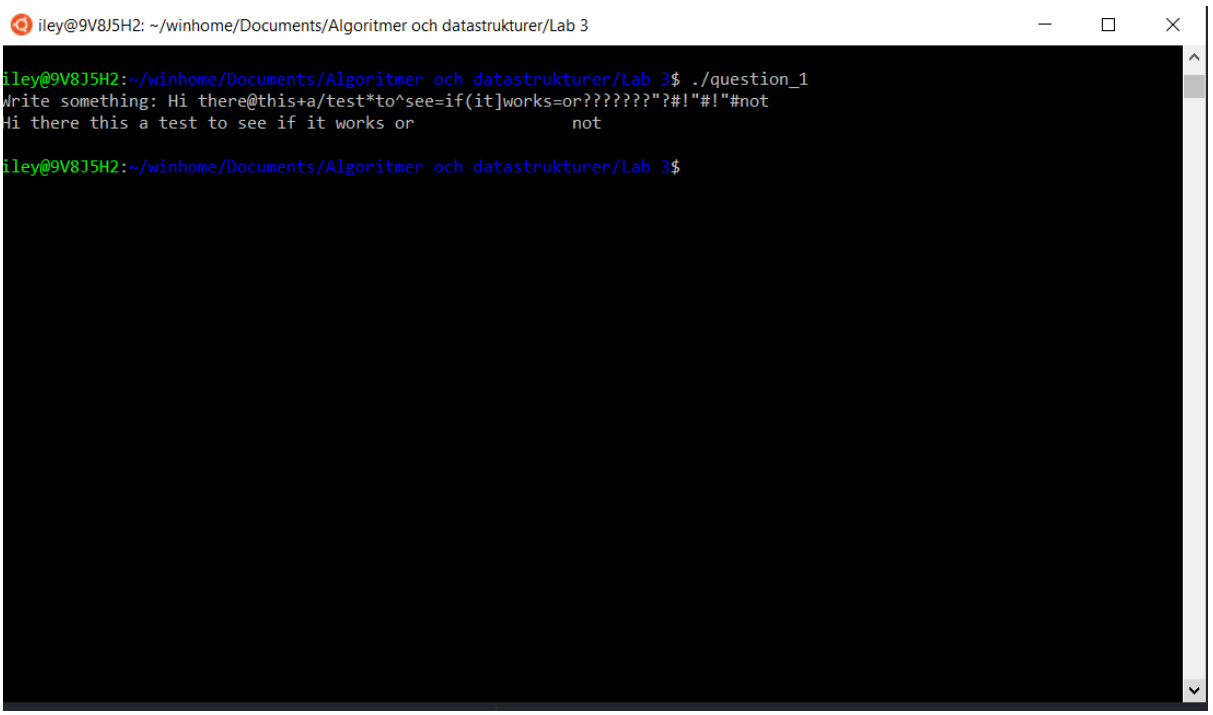
The difference in the main method compared to lab 2 was mainly that in the end the user gets to write a word and the program will say how many times that word is in the text. This was done by getting the user's input and simply using the get method in the separate chaining hash symbol table that gets how many times the value has been put into the list and then the user gets it printed out. This is done in a loop of course, so the user can do it how many times he/she wants to.

Lab 7

For this Lab I used the same binary search tree as in Lab 3 but I added two methods, one to print out the numbers in order and one in reverse order (both with time complexity O(N) and memory complexity O(log(n)), assuming the tree is balanced). I will start with explaining the one that prints out the keys in alphabetical order. First printAscending is called, this method just calls printAscending again but with the root element of the tree (so the top element that all nodes lead to). In this method first it is checked if the node parameter is null, then we return nothing. If not, the method is called again with the left node of the parameter node, so this happens until we reach the leftmost node. When we reach the leftmost node we print out it's key value. Then the method is called again but with the right value of the parameter. What this does is basically that we go to the far left every time at the beginning, print out the node then go up to the root again and if there is a right before that we print that one out and then continue going up. Lastly we go down the right side from the root element and if the node we pass when going to the most right has a left node we print that one out and keep going left if possible. And lastly we go up again from the rightmost element and at the end we end up at the root element again and all nodes have been printed out. The method that prints in reverse alphabetical order is just the reverse method, so we start going down right and then left. This has time complexity N since we simply go through every node and print them out one at a time, there aren't any loops in each other or something that would make it slower or faster. The memory complexity is log(N), if the tree is balanced, since then the memory complexity will be the max height of the tree which is log(N).

Examples of executions

Lab 1 (note: different console since I don't run C in the same way as Java)



Lab 2

```
456 ord: under
457 ord: up
458 ord: updated
459 ord: us
460 ord: use
461 ord: using
462 ord: very
463 ord: view
464 ord: was
465 ord: way
466 ord: we
467 ord: weather
468 ord: went
469 ord: were
470 ord: whatsoever
471 ord: when
472 ord: where
473 ord: which
474 ord: whole
475 ord: whom
476 ord: will
477 ord: winter
478 ord: wisdom
479 ord: with
480 ord: within
481 ord: woods
482 ord: work
483 ord: world
484 ord: worst
485 ord: www
486 ord: yards
487 ord: year
488 ord: years
489 ord: yet
490 ord: you
491 ord: youth
492 ord: ï
max      = the, Value:60
distinct = 492
words    = 1000
Time it took in seconds to find the max value word in the binary search symbol table: 0.291
Time it took in seconds to put in the words into the binary search symbol table: 0.017
Press any key to continue . . .
```

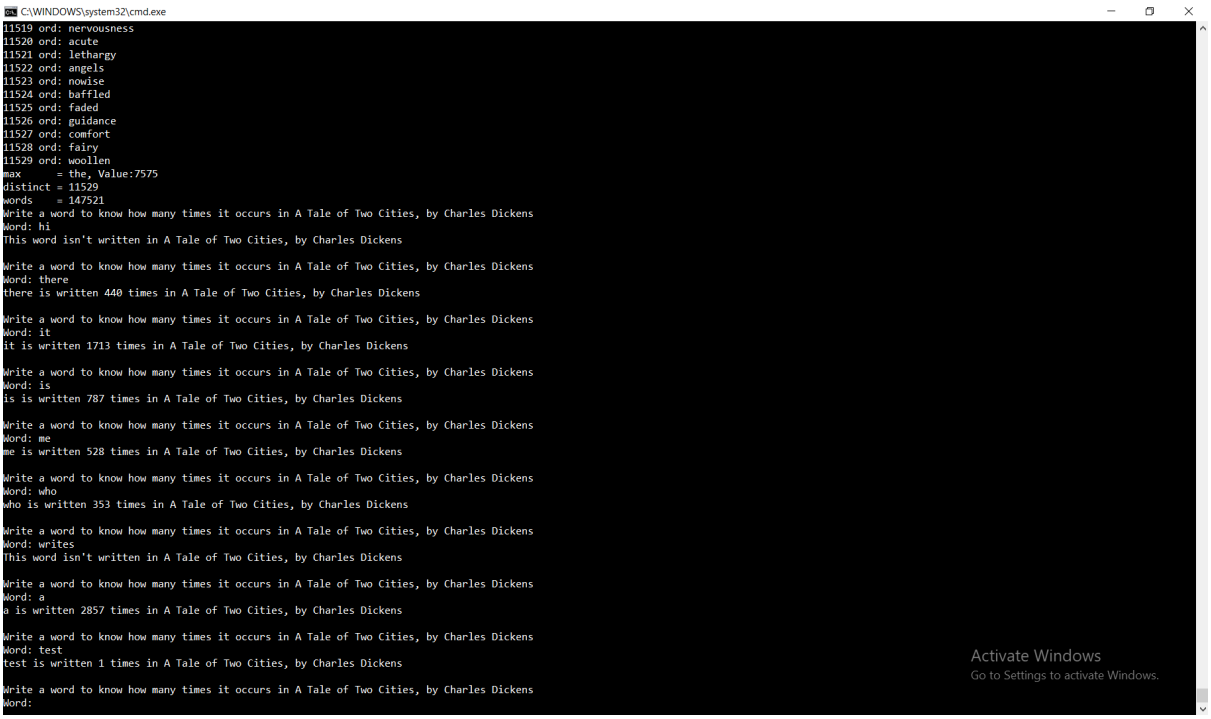
Lab 3

```
469 ord: were
470 ord: whatsoever
471 ord: when
472 ord: where
473 ord: which
474 ord: whole
475 ord: whom
476 ord: will
477 ord: winter
478 ord: wisdom
479 ord: with
480 ord: within
481 ord: woods
482 ord: work
483 ord: world
484 ord: worst
485 ord: www
486 ord: yards
487 ord: year
488 ord: years
489 ord: yet
490 ord: you
491 ord: youth
492 ord: ï
max      = the, Value:60
distinct = 492
words    = 1000
Time it took in seconds to find the max value word in the binary search symbol table: 0.276
Time it took in seconds to put in the words into the binary search symbol table: 0.017
Press any key to continue . . .
```

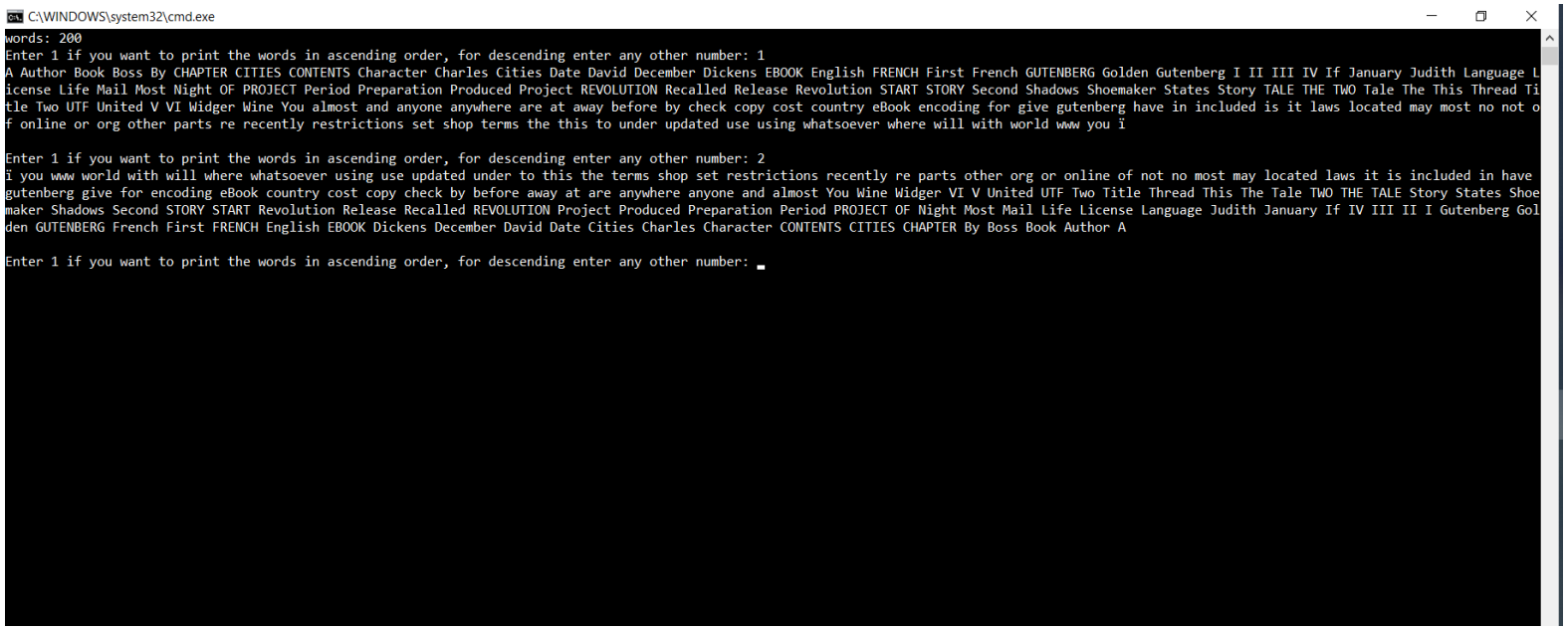
Lab 5

```
max      = the, Value:7575
distinct = 11529
words    = 147521
distinct hash codes = 11527
max hash = 2.147316258E9
min hash = -2.147416853E9
Hash range of numbers: 4.294733111E9
Codes in first tenth of the Hash range: 775.0
Codes in second tenth of the Hash range: 1050.0
Codes in third tenth of the Hash range: 1550.0
Codes in fourth tenth of the Hash range: 919.0
Codes in fifth tenth of the Hash range: 670.0
Codes in sixth tenth of the Hash range: 3707.0
Codes in seventh tenth of the Hash range: 674.0
Codes in eighth tenth of the Hash range: 813.0
Codes in ninght tenth of the Hash range: 633.0
Codes in tenth tenth of the Hash range: 736.0
Press any key to continue . . .
```

Lab 6


C:\WINDOWS\system32\cmd.exe
11519 ord: nervousness
11520 ord: acute
11521 ord: lethargy
11522 ord: angels
11523 ord: nowise
11524 ord: baffled
11525 ord: faded
11526 ord: guidance
11527 ord: comfort
11528 ord: fairy
11529 ord: woollen
max = the, Value:7575
distinct = 11529
words = 147521
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: hi
This word isn't written in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: there
there is written 440 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: it
it is written 1713 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: is
is is written 787 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: me
me is written 528 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: who
who is written 353 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: writes
This word isn't written in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: a
a is written 2857 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word: test
test is written 1 times in A Tale of Two Cities, by Charles Dickens
Write a word to know how many times it occurs in A Tale of Two Cities, by Charles Dickens
Word:
Activate Windows
Go to Settings to activate Windows.

Lab 7


C:\WINDOWS\system32\cmd.exe
words: 200
Enter 1 if you want to print the words in ascending order, for descending enter any other number: 1
A Author Book Boss By CHAPTER CITIES CONTENTS Character Charles Cities Date David December Dickens EBOOK English FRENCH First French GUTENBERG Golden Gutenberg I II III IV If January Judith Language L
icense Life Mail Most Night OF PROJECT Period Preparation Produced Project REVOLUTION Recalled Release Revolution START STORY Second Shadows Shoemaker States Story TALE THE TWO Tale The This Thread Ti
tle Two UTF United V VI Widger Wine You almost and anyone anywhere are at away before by check copy cost country eBook encoding for give gutenberg have in included is it laws located may most no not o
f online or org other parts re recently restrictions set shop terms the this to under updated use using whatsoever where will with world www you ï
Enter 1 if you want to print the words in ascending order, for descending enter any other number: 2
ï you www world with will where whatsoever using use updated under to this the terms shop set restrictions recently re parts other org or online of not no most may located laws it is included in have
gutenberg give for encoding eBook country cost copy check by before away at are anywhere anyone and almost You Wine Widger VI V United UTF Two Title Thread This The Tale TWO THE TALE Story States Shoe
maker Shadows Second STORY START Revolution Release Recalled REVOLUTION Project Produced Preparation Period PROJECT OF Night Most Mail Life License Language Judith January If IV III II I Gutenberg Gol
den GUTENBERG French First FRENCH English EBOOK Dickens December David Date Cities Charles Character CONTENTS CITIES CHAPTER By Boss Book Author A
Enter 1 if you want to print the words in ascending order, for descending enter any other number: