Geraldina Zhang
COP3530
9/21/18

Programming Assignment 1 Commentary

1. getSize() is O(1) because it simply returns the variable "size" in the linked list class. The statement is executed exactly once when the method is called.
   printList() is O(n) because it traverses through the length of the list (while iterator is not equal to null). The number of statements executed in the while loop is proportional to the number of nodes in the list.
   searchList(string searchString) is O(n) because it also travels the length of the list, using a while loop to go to every node, to check if the value at the node contains the string we are searching for.
   editNode(string editString, int index) is O(n) because in the worst-case scenario, the node we are editing is at the end of the linked list. This means that we must iterate through the entire list, and the time it takes for the method to execute grows linearly as the size of the list increases. The for loop shows that the iterator is moved to the next node n times.
   insert(string inputString, int index) is O(n) because we take the slowest of the sequence of statements of the if-else block. The code inside the if statement is executed in constant time and does not depend on the input; however, the code for the else statement has a for loop. The for loop executes its statement once for variable index, or n, number of times. Therefore, max(O(1), O(n)) is O(n).
   deleteNode(int index) is O(n) for reasons similar to the insert function. The complexity of the else statements is greater than that of the if statements, so we take the former as our overall computational complexity.
   My line editor is implemented inside my main() function. Its complexity is O(n) because the else if statement(s) with the largest complexity is O(n + n) = O(2n) = O(n). For example, when the command is identified as "insert", we find all integers before the start of a quotation mark (in order to parse for the index) using a for loop. The code within the for loop is executed n times. Then, there is another block of code to actually call the insert() method, which we determined was also O(n) above. We add the two complexities to get O(2n), which is then simplified to O(n).
2. One of the advantages of using a linked list to implement a line editor is the fact that we are able to dynamically allocate memory as we need it. I did not have to initialize a size beforehand and risk having unused/wasted memory. Additionally, inserting and deleting nodes is easy to do with pointers, versus shifting everything as we would in an array-based list (an expensive operation). All I had to do was create new references or break old ones. A disadvantage would be the fact that we cannot access elements randomly like we can in an array. For functions that take in an index such as insert, edit, and delete, the time to access a node (especially towards the end of a list) is longer since we can only move in one direction from one node to another.
3. This assignment reinforced the importance of debugging. Since I couldn't see "under the hood" what was happening inside my linked list, writing code for outputs within my iterating for loops or while loops helped me visualize where my pointers were. My foundation in C++ was not very strong coming into this class, but writing my own methods for the linked list solidified the concept of pointers for me (something that I

always had trouble with!). I was forced to draw the nodes and pointers on paper and manipulate them to achieve my goal. By spending time on this kind of data structure from a low level, I really got to understand the process of insertions, deletions, etc. If I had to start over, I would have my line editor implementation as a separate method so that my main() looked cleaner. I would also try and reduce my complexity score. Lastly, I would have liked to implement try-catch blocks for errors, like accessing out of bound indexes, rather than ignoring them.