1. The Linked List class has eight primary methods associated with it. These methods and their time complexities are listed below.

`void insertEnd(std::string newLine)`	This method attaches a node to the end of the linked list. If there is no linked-list to attach a node to, it will create a new list. It's time complexity is O(1).
`void insertEnd(std::string newLine)`	This method attaches a node to the end of the linked list. If there is no linked-list to attach a node to, it will create a new list. It's time complexity is O(1).
`void insertFront(std::string newLine)`	This method attaches a node to the front of the list. It's time complexity is O(1).
`Node* nodeAt(int index)`	This method returns the Node at a specified index. This list is doubly-linked, so the worst-case scenario is twice as good as that of a singly-linked list. The time complexity, however, is still O(n).
`void insertAt(int index, std::string newLine)`	This method inserts a node at a specified index. The insertion itself is an O(1) process.
`void search(const std::string &toSearch)`	This method searches for a string within the linked list equal to its parameter. Because the position is unknown, it must iterate through every element in the list, beginning at the head. The time complexity is therefore O(1).
`void edit(int index, std::string newLine)`	This method changes the value of a node at a specified index. The time complexity is O(1). This is possible because the pointer to the indexed node is available.
`void deleteNode(int index)`	Like the insertAt() method, this method is has a complexity of O(1). This is possible because the pointer to the indexed node is available.
`void printList()`	This method must iterate through every node in the linked list and print its value. Therefore, the time complexity is O(n).

The remainder of the methods in the class have a time complexity of O(1).

2. I believe that using a linked list for a line editor was a good decision. This line editor can be amended many times over without having to reallocate memory. This is a very expensive process (computationally) that arrays have to deal with when inserting and deleting data. Searches are of equal complexity across the board. However, arrays do have the benefit of accessing data much faster than linked lists (O(1) vs O(n)). This allows for faster information retrieval times.

3. Initially, I had a strong aversion towards linked lists. In retrospect, this is because I did not have much experience implementing them- they were a scary concept. By the time I finished the assignment, I learned that they are, in many ways, much more diverse than arrays in some ways (discussed in response #2). I, do, however, not using templates for the project. I understand the value behind generic types and would have liked to have been able to create a more capable linked list.