The time complexities of each of the functions in my program are as follows:

- insertEnd: O(n) – Creation of the new node is constant time, however, this algorithm has to iterate to the end of the list to insert. I originally had an implementation with a tail node but the memory allocations had different time responses stepik so the tail node was deleted before it could be reallocated.
- search: O(n) – Because the search algorithm is guaranteed to iterate through the entire list in order to capture every single instance of the string it is looking for it must be O(n) time. It is not higher because there are no other loops inside of the traversal loop, and all of the other operations in the function are constant time.
- print: O(n) – Similar to the search function, the print function must iterate through the entire linked list and display the value at each node. The traversal through the list is O(n) and the accessing of node data for each iteration is constant time, which simplifies to O(n).
- iterateTo: O(n) – Function I created to iterate to specific index I am looking for. Was created to avoid repeated code. Technically O(n – 1) to provide previous node to one action will be applied to, which is no different than O(n).
- delete: O(n) – All node creation and pointer reassignments are constant time, however, the function calls on iterateTo, therefore it becomes O(n).
- insert: O(n) – Function calls on iterateTo so that it can insert a node at that specific index. Insertion is constant time, however, traversal is O(n).
- edit: O(n) – Similar to insert this function calls on iterateTo and must traverse to a specific node to change it. The changing of data at the node is constant time, but the traversal, once again, costs O(n) time.
- main: At worst main is O(n + k). This occurs in instances where search, insert, and edit are called. The n in the Big O notation signifies the traversal of the linked list and the k signifies the iteration through the keyboard input to find the start of the string to be used in a node. Because these sizes are not the same they must be different variables. Though, the running time remains linear.

A linked list for this project is the most likely the most optimal solution because of the insertion and deletion features from anywhere in the list. Linked lists provide O(n) run time for insertions and deletions from anywhere in the list because the actual deletion or insertion and pointer reallocation for each procedure provides constant time operations, and traversal costs O(n). Meanwhile, In an array, although the traversal is faster because the array is contiguous, if an element is deleted from anywhere other than near the end of the list, there is a significant cost to keep the memory addresses contiguous and reallocate every memory location following the insertion or deletion. Furthermore, the array would have to be initialized to a constant size, causing problems if the number of inserted values surpasses the limit. This could be solved by implementing an arraylist, however this solution still suffers the insertion and deletion problems discussed above. There are few disadvantages of a linkedlist for this project, however, one might be that if there are few insertions or deletions from the middle of the list and more searches, edits, or prints, an alternative solution could work better. This is because the traversal and accessing of nodes in a linked list is slower than something like an arraylist. Another data structure that could provide better time would be a binary tree. An AVL sorted tree with nodes that contained an index value that the nodes were sorted by could be a fast implementation as well. The search, deletion, and insertion times are O(log n) and sorting is constant time, which makes this a particularly fast implementation for functionality like this program. Despite this drawback a linkedlist could be the best implementation for this project based on the functions included and the number of nodes inputted to the list.

I learned a couple of things from this project: the first being that different compilers can react to the same code in different ways, and I became more familiar with string manipulation in C++. My original implementation used a tail node that allowed for the insertEnd method to have an O(1) run time, however, I ended up having to change it to a traversal of the list then insertion. I also became more familiar with how to select certain parts of a string and read input in C++. In most other projects cin is used to read singular input but for the line editor it took some creativity to select certain parts following the function call and then input the correct strings and integers into the functions. If I had to do this project again I would spend more time implementing the tail node in stepik so that it worked properly.

```
#include <iostream>
#include <sstream>

struct node
{
   std::string line;
   node* next;
};

class LinkedList
{
private:
   node* head;
```

```
public:
   LinkedList()
   {
      head = nullptr;
   }
   void insertEnd(std::string addString);
   std::string search(std::string findThis);
   void print();
   void insert(int index, std::string line);
   node* iterateTo(int index);
   void deleter(int index);
   void edit(int index, std::string changeString);
};

void LinkedList::insertEnd(std::string addString)
{
   node* temp = new node();
   temp->line = addString;
   temp->next = nullptr;
   if(head == nullptr)
   {
      head = temp;
   }
   else
   {
      node* curr = head;
      while(curr->next != nullptr)
      {
         curr = curr->next;
      }
      curr->next = temp;
   }
}

std::string LinkedList::search(std::string findThis)
{
   node* curr = head;
   int i = 0;
   std::string found;
   while(curr != nullptr)
   {
      ++i;
      if(curr->line.find(findThis) != std::string::npos || curr->line == findThis)
      {
            found.append(std::to_string(i) + " " + curr->line + "\n");
        }
        curr = curr->next;
    }
    if(found.empty())
    {
        found = "not found\n";
    }
    found = found.substr(0, found.length() - 1);
    return found;
}

void LinkedList::print()
{
    node* curr = head;
    int i = 1;
    while(curr != nullptr)
    {
        std::cout << i << " " << curr->line << std::endl;
        curr = curr->next;
        ++i;
    }
}

node* LinkedList::iterateTo(int index)
{
    node* prev = head;
    for(int i = 1; i < (index - 1); ++i)
    {
        if(prev->next != nullptr)
        {
            prev = prev->next;
        }
        else
        {
            return nullptr;
        }
    }
    return prev;
}

void LinkedList::deleter(int index)
{
    if(index == 1)
    {
        node* temp = head;
        head = temp->next;
        delete temp;
        return;
    }
    node* prev = iterateTo(index);
    if(prev == nullptr || prev->next == nullptr)
    {
        return;
    }
    node* curr = prev->next;
    prev->next = curr->next;
    delete curr;
}

void LinkedList::insert(int index, std::string line)
{
    node* newNode = new node;
    newNode->line = line;
    if(index == 1)
    {
        newNode->next = head;
        head = newNode;
        return;
    }
    node* prev = iterateTo(index);
    if(prev == nullptr)
    {
        return;
    }
    newNode->next = prev->next;
    prev->next = newNode;
}

void LinkedList::edit(int index, std::string changeString)
{
    if(head == nullptr)
    {
        return;
    }
    if(index == 1 && head != nullptr)
    {
        head->line.clear();
        head->line = changeString;
        return;
    }
    node* prev = iterateTo(index)->next;
    if(prev != nullptr)
    {
        prev->line.clear();
        prev->line = changeString;
    }
    return;
}

int main() {
    LinkedList* list = new LinkedList();

    std::string indexStr;
    int indexInt;
    std::string command;
    std::string quote;
    while(true)
    {
        std::cin >> command;
        if(command == "insertEnd" || command == "search")
        {
            quote.clear();
            getline(std::cin, quote);
            if(quote[quote.length() - 1] == '"')
            {
                for(int i = 0; i < quote.length(); ++i)
                {
                    if(quote[i] == '"')
                    {
                        if(i != (quote.length() - 1))
                        {
                            quote = quote.substr(i + 1, quote.length() - 3);
                            break;
                        }
                        else
                        {
                            continue;
                        }
                    }
                }
                if(quote.find("\"") == std::string::npos)
                {
                    if(command == "search")
                    {
                        std::cout << list->search(quote) << std::endl;
                        continue;
                    }
                    list->insertEnd(quote);
                }
            }
            continue;
        }
        else if (command == "quit")
        {
            break;
        }
        else if(command == "print")
        {
            list->print();
        }
        else if (command == "insert" || command == "edit")
        {
            std::cin >> indexStr;
            std::stringstream(indexStr) >> indexInt;

            quote.clear();
            getline(std::cin, quote);
            if(quote[quote.length() - 1] == '"')
            {
                for(int i = 0; i < quote.length(); ++i)
                {
                    if(quote[i] == '"')
                    {
                        if(i != quote.length() - 1)
                        {
                            quote = quote.substr(i + 1, quote.length() - 3);
                            break;
                        }
                        else
                        {
                            continue;
                        }
                    }
                }if(quote.find("\"") == std::string::npos)
                {
                       if(command == "insert")
                       {
                          list->insert(indexInt, quote);
                          continue;
                       }
                       list->edit(indexInt, quote);
                    }
                 }
              }
              else if(command == "delete")
              {
                 std::cin >> indexStr;
                 std::stringstream(indexStr) >> indexInt;
                 list->deleter(indexInt);
              }
              else
              {
                 continue;
              }
           }
        }
```