Lab Assignment #3

Linked Lists

Name: Bhavyai Gupta
UCID: 30143691

Algorithm for checking anagrams

We use the method `compare()` defined in the class `CharCount` to check if the two strings are anagrams or not.

The algorithm works by comparing the count of each letter of one string with the count of each letter of another string. If the strings are anagrams of each other, this count must match for all the letters.

Below is the implementation of the method.

```
/**
 * Compare if other CharCount object c is equal to this
 *
 * @param c other CharChount object to be compared
 * @return <code>true</code> if this and c have same letters with same count,
 *         <code>false</code> otherwise
 */
public boolean compare(CharCount c) {
    // loop to check every alphabetic character in the array chars of this and c
    for (int i = 0; i < this.chars.length; i++) {
        if (this.chars[i] != c.chars[i]) {
            return false;
        }
    }

    return true;
}
```

This count of each letter is stored in an array called `chars`, which is created using the `CharCount` constructor. Please refer to the code below for its implementation.

```java
/**
 * Creates a CharCount object and break down the String s into integer array
 *
 * @param s String
 */
public CharCount(String s) {
    this.chars = new int[(int) 'z' + 1];

    for (int i = 0; i < s.length(); i++) {
        this.chars[(int) s.charAt(i)] = this.chars[(int) s.charAt(i)] + 1;
    }
}
```

Complexity Analysis

1. Complexity of CharCount constructor

The for-loop runs exactly **k** times, where **k** is the number of letters in a word.

Time Complexity = Θ(k).

2. Complexity of method compare

The worst case happens when both the strings are anagrams of each other. In such a case, the for-loop runs maximum **n** times, where **n** is the length of the array `chars` we created. **n** is always equal to 123 in our implementation.

Time Complexity = O(n)

3. Overall complexity to determine anagrams

Time Complexity = O(n + k)