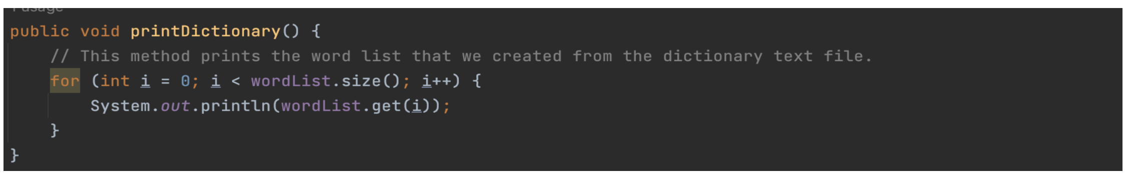
ANNANYA JAIN (CS 284 - C)

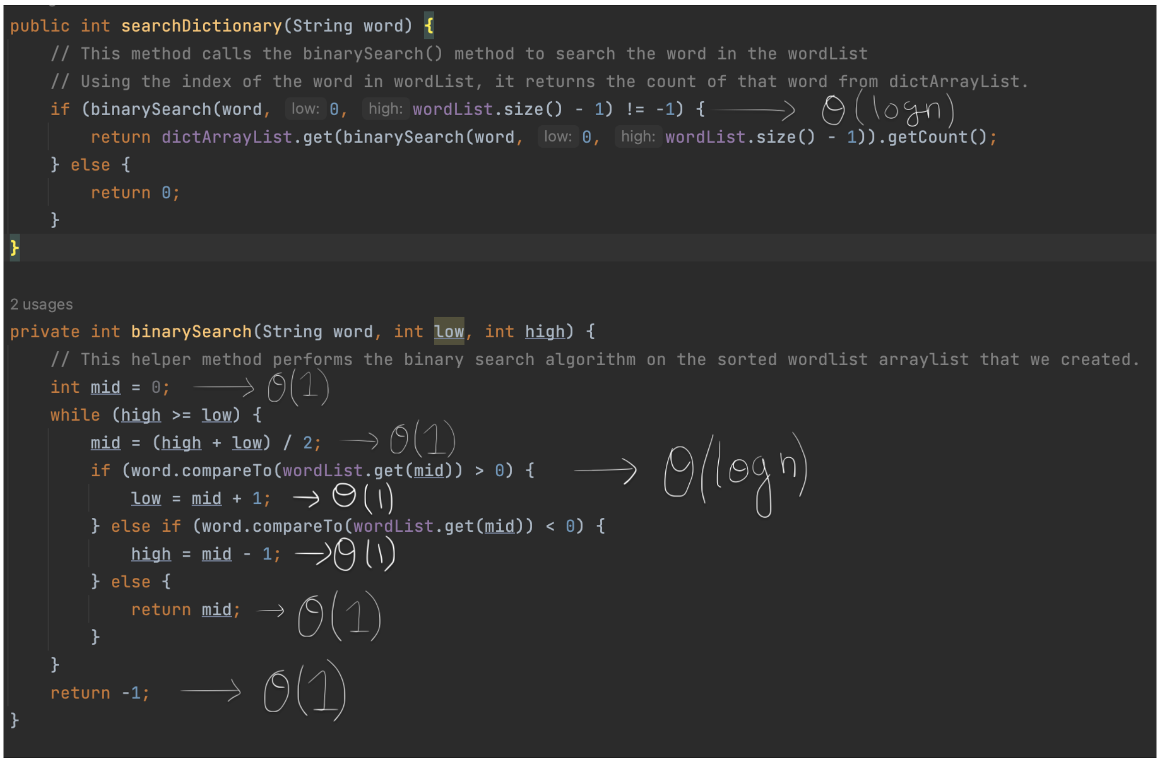
I pledge my honor that I have abided by the Stevens Honor System.

RUNTIME ANALYSIS



```java
public void printDictionary() {
    // This method prints the word list that we created from the dictionary text file.
    for (int i = 0; i < wordList.size(); i++) {
        System.out.println(wordList.get(i));
    }
}
```

In **printDictionary()**, there is a for loop where variable i is initialised with value 1. The for loop starts with i = 0 and iterates till it reaches the size of the wordlist (an arrayList). In each iteration, i is incremented by 1. Since there are no nested loops or function calls, printDictionary() will have **linear time complexity** and hence its Big-O Notation is **O(n).**



```java
public int searchDictionary(String word) {
    // This method calls the binarySearch() method to search the word in the wordList
    // Using the index of the word in wordList, it returns the count of that word from dictArrayList.
    if (binarySearch(word,  low: 0,  high: wordList.size() - 1) != -1) {
        return dictArrayList.get(binarySearch(word,  low: 0,  high: wordList.size() - 1)).getCount();
    } else {
        return 0;
    }
}

2 usages
private int binarySearch(String word, int low, int high) {
    // This helper method performs the binary search algorithm on the sorted wordlist arraylist that we created.
    int mid = 0;
    while (high >= low) {
        mid = (high + low) / 2;
        if (word.compareTo(wordList.get(mid)) > 0) {
            low = mid + 1;
        } else if (word.compareTo(wordList.get(mid)) < 0) {
            high = mid - 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

In **searchDictionary()**, there is an if-else statement but since it calls a helper function named binarySearch, it will have the runtime complexity of binarySearch function.

binarySearch() is quite efficient in finding an element within a sorted list. During each iteration, binary search reduces the search space by half. The search terminates when the element is found. It first checks that element is on which side (i.e left or right) of the middle element in the sorted list. It then again divides that part of the list by 2 where the element could be found and reassigns the low and high value. Thus, this has logarithmic runtime complexity and its Big - O Notation is O(log n). The search space is reduced by a factor of 2 at each iteration, so the **number of iterations required to find the target element is proportional to log n.** The other statements as shown in the code, will have constant runtime i.e O(1).

Hence, **searchDictionary()** will have **logarithmic runtime complexity i.e Big-O Notation is O(log n)**, i.e same as binarySearch().