Anthony Curcio-Petraccoro
Dr. Akcam-Kibis
CS 284 - Section A
2/26/2023

```
public void printDictionary() {

    for (int i = 0; i < wordList.size(); i ++) {

        System.out.println(wordList.get(i));

    }

}
```

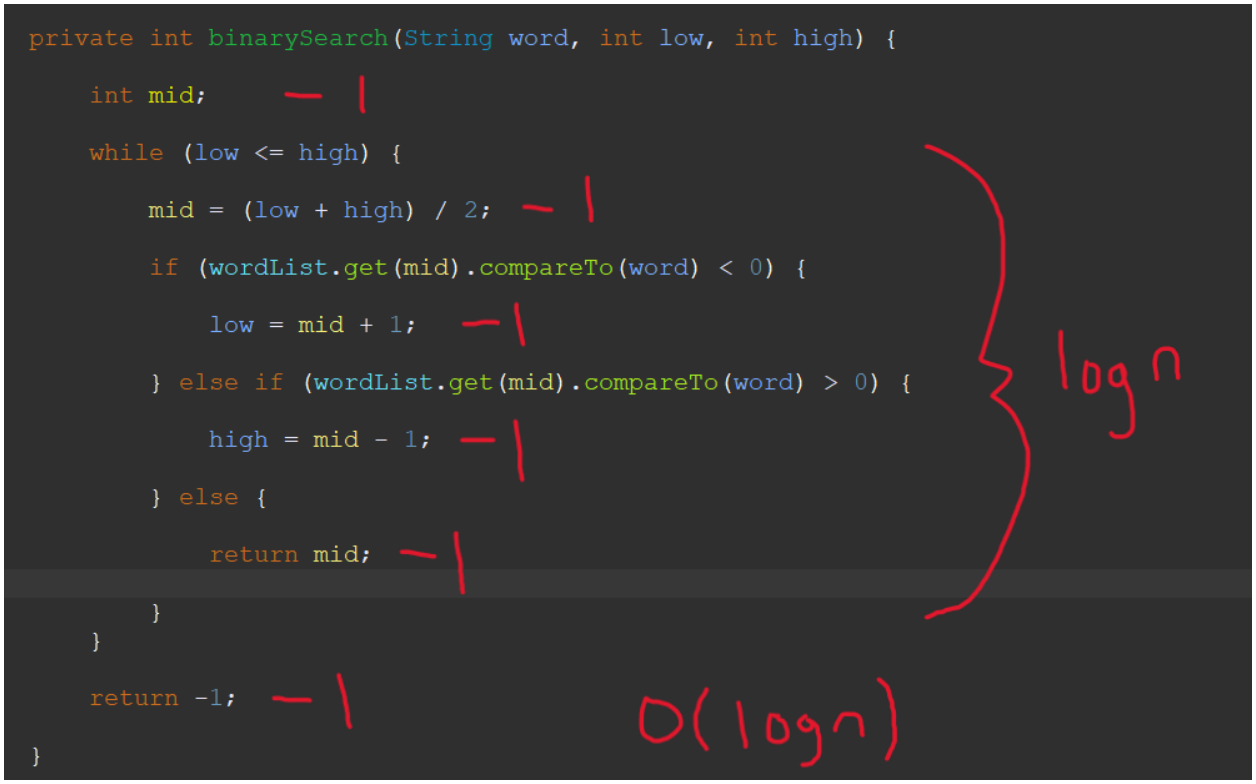
O(n)

This method only requires one for loop, incrementing linearly. The for loop will run depending on the size of the array “wordList.” Since the size of “wordList” can change depending on the number of words it contains, the loop has a runtime of O(n), but we know the size of wordList in this case. Since the for loop’s run time is larger than the run time of the single print statement (which has a run time of a constant), the Big O notation for the entirety of the method is O(n).

```
public int searchDictionary(String word) {

    int index = binarySearch(word, 0, wordList.size());
    int counter = -1;

    if (index == -1) {

        System.out.print("The word '" + word + "' does not exist in the Ion dictionary!\r\n");

    }

    if (index >= 0) {

        counter = dicArrayList.get(index).getCount();
        System.out.println("The word '" + word + "' occured " + counter + " time(s) in the book!");

    }

        return counter;

}
```

O(logn)

The searchDictionary method has a runtime of O(logn). The returns and assignment statements have a constant runtime. The one assignment statement that has a runtime greater than a constant is the assignment of “index”. Since the assignment of “index” calls binarySearch and requires an output, the runtime of the assignment of “index” will be the same as binarySearch, which is O(logn). Since the assignment of “index” has the largest runtime in the method, the runtime of the entire seachDictionary method is O(logn).



The binary search method has a runtime of O(logn). The function of binary search is to target the midpoint of the array and compare it to the desired element. Whichever half of the array the target item is said to be less than or greater than, the function will then hone in on that half of the array, eliminating the other half from the search. After each iteration of the while loop in binarySearch, the size of the array will be cut in half (i.e. iteration 1 - n; iteration 2 - n/2; iteration 3 - n/4, …, with n being the size of the array) until only the target element remains. In mathematical terms, you want to find the constant “x” such that n/2^x is less than or equal to 1, with 1 being the final size of the array. After algebraic manipulation, you are left with 2^x greater than or equal to n, which is equal to log2(n). The thing to keep in mind when implementing binary search is that the array being searched must be in order. If the array was not in order, the algorithm won’t be able to split the array in half and successfully find the proper element, since the sole reason binary search works is because the elements of the array are in order. The various other returns and assignment statements have a constant runtime. Since the binary search algorithm has a runtime of O(logn), which is the largest runtime of the function, the entire binarySearch method has a runtime of O(logn).