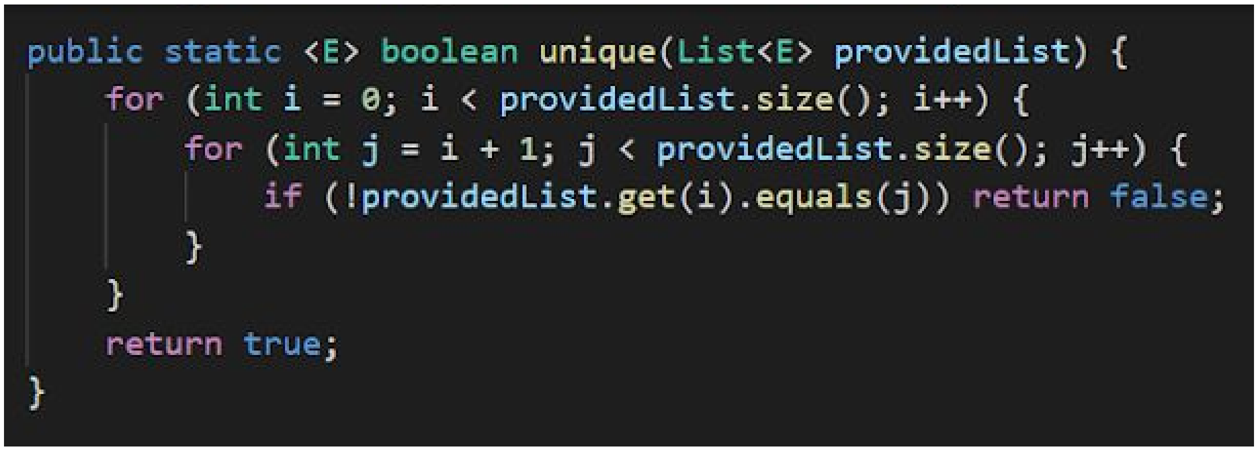
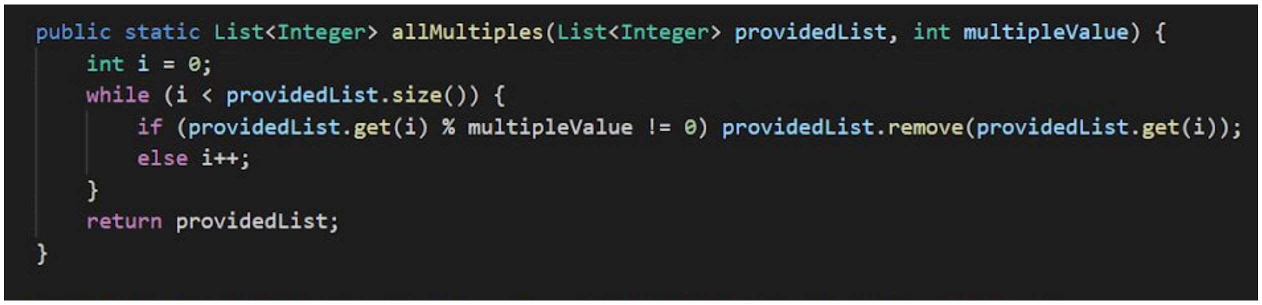
1) O(n^2)
2) O(n^4)
3) O(n^2)
4) O(n^3)
5) O(1)
6) O(1)
7) O(n)
8) O(n)
9) O(nlog(n))
10) O(n^3)
11) O(1)
12) O(n)
13) When you remove an item from the middle of a list, you need to adjust the indexes of all the numbers after it. This would have to be done n - <item to remove's index> - 1 times which in big O notation is O(n).
14) O(1)
15) If you do not have to reallocate, you get the best time of O(1). If you do have to reallocate you get O(n).
16) An arraylist would not be good if you had a constantly growing list that you did not know what size it was going to be because as it grows it as to reallocate and takes longer. If you are not adding a lot of items or if you know how many items you are adding (because then you can initialize the arraylist to be the proper size) then an arraylist is a good choice.
17) O(n)
18) O(n)
19) O(n^2)
20) Infinity
21) Bogosort just keeps randomizing the order so the list is never guaranteed to become closer to being sorted or operate in any pattern. Theoretically it could keep randomizing and never become sorted.
22) O((n+1)!)
23) Since it is randomized, the odds of being a sorted list are the same as the probability of picking a deck of cards in a specific order (52*51*50*...*1 aka n!). However when you shuffle, you must spend n to do the shuffle. Therefore it is really n * n! which is (n+1)!
24)



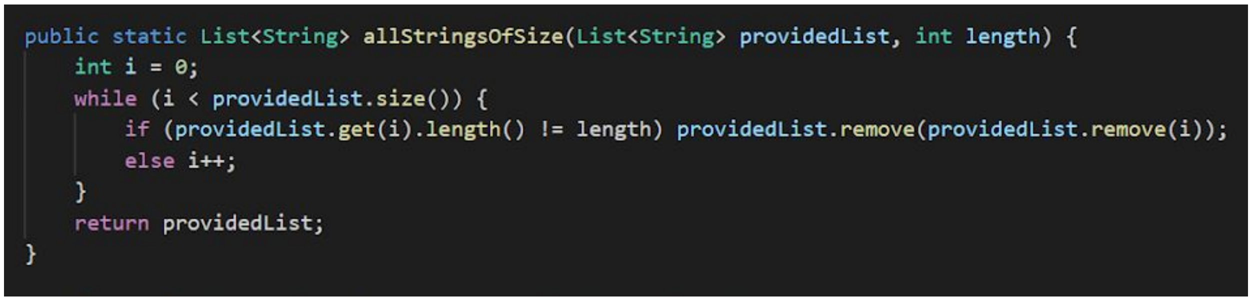
```java
public static <E> boolean unique(List<E> providedList) {
    for (int i = 0; i < providedList.size(); i++) {
        for (int j = i + 1; j < providedList.size(); j++) {
            if (!providedList.get(i).equals(j)) return false;
        }
    }
    return true;
}
```

Unique method: O(n^2)



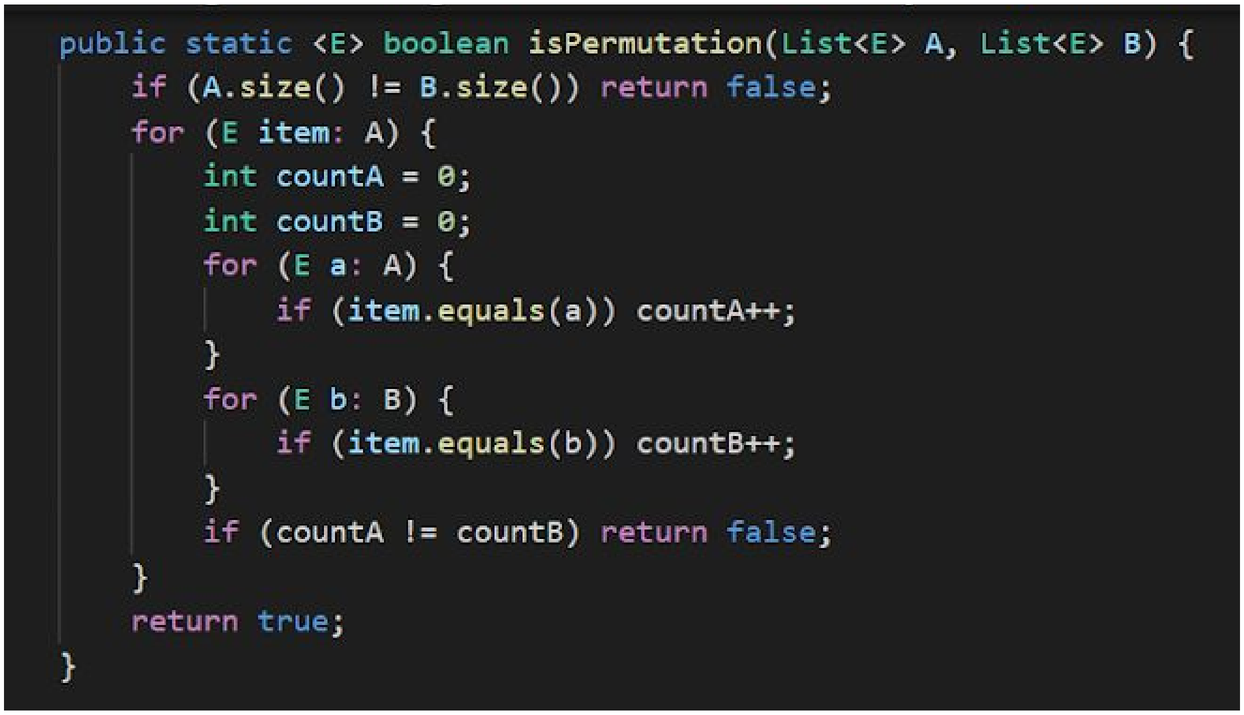
```java
public static List<Integer> allMultiples(List<Integer> providedList, int multipleValue) {
    int i = 0;
    while (i < providedList.size()) {
        if (providedList.get(i) % multipleValue != 0) providedList.remove(providedList.get(i));
        else i++;
    }
    return providedList;
}
```

All Multiples method: O(n^2)



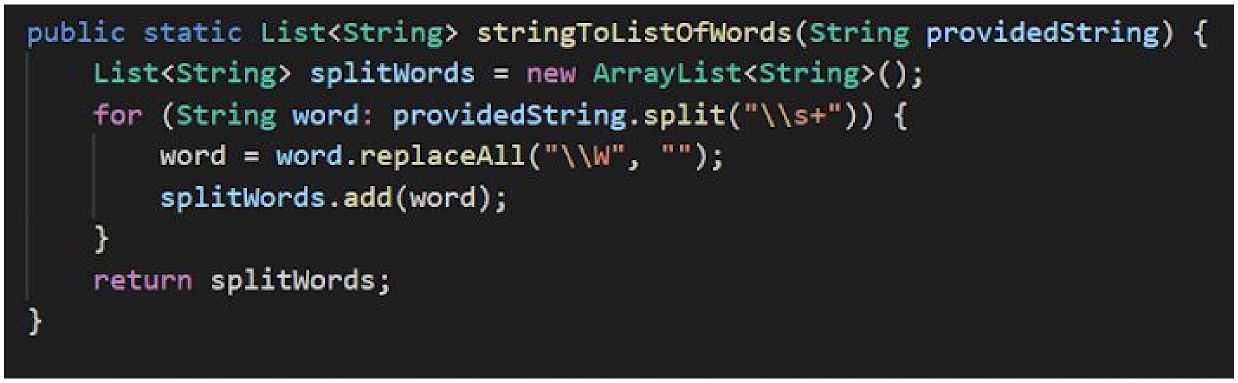
```java
public static List<String> allStringsOfSize(List<String> providedList, int length) {
    int i = 0;
    while (i < providedList.size()) {
        if (providedList.get(i).length() != length) providedList.remove(providedList.remove(i));
        else i++;
    }
    return providedList;
}
```

All Strings of Size method: O(n^2)



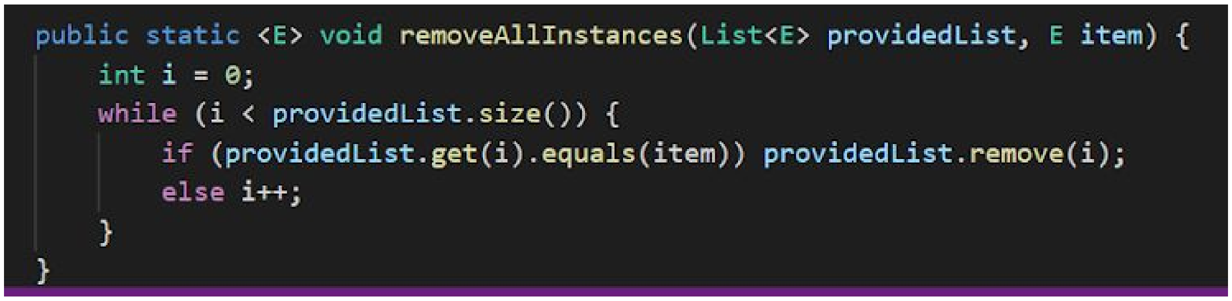
```java
public static <E> boolean isPermutation(List<E> A, List<E> B) {
    if (A.size() != B.size()) return false;
    for (E item: A) {
        int countA = 0;
        int countB = 0;
        for (E a: A) {
            if (item.equals(a)) countA++;
        }
        for (E b: B) {
            if (item.equals(b)) countB++;
        }
        if (countA != countB) return false;
    }
    return true;
}
```

Is Permutation method: $O(n_a(n_a+n_b))$



```java
public static List<String> stringToListOfWords(String providedString) {
    List<String> splitWords = new ArrayList<String>();
    for (String word: providedString.split("\\s+")) {
        word = word.replaceAll("\\W", "");
        splitWords.add(word);
    }
    return splitWords;
}
```

String To List Of Words method: O(n^2)



```java
public static <E> void removeAllInstances(List<E> providedList, E item) {
    int i = 0;
    while (i < providedList.size()) {
        if (providedList.get(i).equals(item)) providedList.remove(i);
        else i++;
    }
}
```

Remove All Instances method: O(n^2)