The questions are all based on **MyArrayListBible** (attached). See the API for **ArrayList** for the complexity of some of its methods, but assume that **add** takes constant time since that is its amortized cost. Assume that the Bible has *n* verses. Finally, assume the methods on Strings take constant time.

For questions 2-7, give tight bounds on the worst-case complexity of the given method. Briefly justify your answer in each case. Question 1 has been done for you to give you an idea of how to justify your answers.

1. public MyArrayListBible(VerseList verses)
 The first two lines take constant time.

The third line takes *Θ(n)* times since the constructor has to copy all of the elements.
The fourth line takes constant time.
Thus, the overall complexity is *Θ(n)+k = Θ(n).*

1. public int getNumberOfVerses()

1. public VerseList getAllVerses()

1. public boolean isValid(Reference ref)

1. public Verse getVerse(Reference r)
2. public VerseList getReferencesContaining(String phrase)

1. public VerseList getVerses(ArrayList<Reference> references)
(Assume references.size() is *m*)
2. One of the previous 7 methods is not very efficient.[[1]](#footnote-1) State which one, explain how to fix it and tell what the complexity of the fixed method would be.

*There are 9 questions on this assignment—****don’t skip question 9 on the last page!***

**public** **class** MyArrayListBible {

 **private** String version;

 **private** String title;

 **private** ArrayList<Verse> theVerses;

 **public** MyArrayListBible(VerseList verses) {

 **this**.version = verses.getVersion();

 **this**.title = verses.getDescription();

 theVerses = **new** ArrayList<Verse>(verses);

 theVerses.add(**new** Verse(BookOfBible.*Dummy*, 1, 1, ""));

 }

 **public** **int** getNumberOfVerses() {

 **return** theVerses.size() - 1;

 }

 **public** VerseList getAllVerses() {

 ArrayList<Verse> verses = **new** ArrayList<Verse>(theVerses);

 verses.remove(**new** Verse(BookOfBible.*Dummy*, 1, 1, ""));

 **return** **new** VerseList(version, title, verses);

 }

 **public** **boolean** isValid(Reference ref) {

 **for** (Verse v : theVerses) {

 **if** (v.getReference().equals(ref)) {

 **return** **true**;

 }

 }

 **return** **false**;

 }

 **public** Verse getVerse(Reference r) {

 **for** (Verse v : theVerses) {

 **if** (isValid(r)) {

 **if** (v.getReference().equals(r)) {

 **return** v;

 }

 }

 }

 **return** **null**;

 }

 **public** ArrayList<Reference> getReferencesContaining(String phrase) {

 ArrayList<Reference> results = **new** ArrayList<Reference>();

 **if** (phrase.length() == 0) {

 **return** results;

 }

 String lowerCasePhrase = phrase.toLowerCase();

 **for** (Verse v : theVerses) {

 **if** (v.getText().toLowerCase().contains(lowerCasePhrase)) {

 results.add(v.getReference());

 }

 }

 **return** results;

 }

 **public** VerseList getVerses(ArrayList<Reference> references) {

 VerseList results = **new** VerseList(version, "");

 **for** (Reference ref : references) {

 results.add(getVerse(ref));

 }

 **return** results;

 }

 **public** ArrayList<Reference> getReferencesInclusive(Reference firstVerse, Reference lastVerse)

 {

 ArrayList<Reference> results = **new** ArrayList<Reference>();

 **if** (firstVerse.compareTo(lastVerse) > 0) {

 **return** results;

 }

 **if** (!isValid(firstVerse) || !isValid(lastVerse)) {

 **return** results;

 }

 **int** index1 = 0;

 **while** (!theVerses.get(index1).getReference().equals(firstVerse)) {

 index1++;

 }

 **int** index2 = 0;

 **while** (!theVerses.get(index2).getReference().equals(lastVerse)) {

 index2++;

 }

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

 **if** (i >= index1 && i <= index2) {

 results.add(theVerses.get(i).getReference());

 }

 }

 **return** results;

 }

1. For the method getReferencesInclusive (above) assume there are *k* references between firstVerse and lastVerse and that firstVerse is the *m*th verse in the array.
	1. About how many operations does getReferencesInclusive take in terms of *n*, *m*, and *k*? Include constants in your answer (e.g. *3n+2m*). Justify your answer.
	2. Rewrite the method so that it is as efficient as possible. *Type, print, and attach your solution.*
	3. About how many operations does your version of getReferencesInclusive take in terms of *n*, *m*, and *k*? As above, include constants and justify your answer. (Hint: It is not necessarily the case that your answer depends on all of *n*, *m*, and *k*.)
1. Actually several are inefficient, but the cause for all of them is a single method. That is the one you should consider. [↑](#footnote-ref-1)