2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
```

GO ON TO THE NEXT PAGE

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call	After the call	
wordList	wordList	numWords
"apple" "berry" "quince"	"apple" "berry" "pear" "quince"	4
"apple" "berry" "pear"	"apple" "berry" "pear"	3
"apple" "fig" "peach"	"apple" "fig" "peach" "pear"	4
"quince" "raisin" "tart"	"pear" "quince" "raisin" "tart"	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
```