IN1002 Coursework

This task will require you to produce Java implementations of 4 search algorithms on an ordered collection of strings. The most relevant lectures are weeks 2 and 4. It is due at 5pm on Sunday 15th March 2020, to be submitted via Moodle. It is worth 30% of the module marks. (The final exam accounts for the other 70%.)

Individual task

This is an individual task and your submission will be run through plagiarism checks. If you copy the work of others (either that of fellow students or of a third party), with or without their permission, you will score no marks and further disciplinary action will be taken against you.

Starting point

Download the `Search` NetBeans project from Moodle. The project contains some same data files and four Java files:

interface StringList

This is an interface to a simple array-like collection of strings. You may assume that the strings are in ascending dictionary order, e.g.
"multiple" < "never" < "no" < "nobody" < "other" < "others" < "otter"

Some of the strings may be duplicated. The methods provided are

size()
: returns the number of strings in the collection

get(int i)
: returns the string at position `i` in the collection

You must **not** change anything in this file.

class FileStrings implements StringList

An implementation of `StringList`, initialized from one of the files in the `data` folder. There are also some additional methods for instrumentation that you can use in your testing.
You must **not** change anything in this class.

class Search

A class of methods, whose bodies you have to fill in. This is what you'll be submitting. You can add extra methods if you like, but there is no need for fields. **You must not change the signatures of the methods supplied.**

SearchTest

A main program that you can use for testing the methods you write in the `Search` class. Change this as much as you like. You won't be submitting it; I'll be using my own test program.

The task

You are to implement the following methods. The cost measure will be the number of calls to the `get()` method.

public int longestWord(StringList a)

Return the index of the longest string in the collection. If there are several strings of the same longest length, return the smallest index. The cost should be *n*, where *n* is the size of the list.

public int countUnique(StringList a)

Return the number of unique elements in the list. The cost should be *n*, where *n* is the size of the list.

public int findElement(StringList a, String k)

Return the index of an element in the list that is equal to `k`, or -1 if it is not in the list. using the first version of binary search discussed in week 4. This should cost log(*n*) in the worst case, but less in other cases.

public int countGreaterOrEqual(StringList a, String k)

Return the number of elements in the list that are greater than or equal to `k` in dictionary order. Note that `k` may not be in the list. You should be able to do this with the second version of binary search discussed in week 4. This should cost log(*n*).

You may use standard Java classes. In particular, you may find the following methods of the String class useful: `equals`, `compareTo`.

Note that the `StringList` data structure being used is a collection (rather than an array), and so you will need to use the `get` method to access elements of the array and the `size` method to determine the number of elements in the collection. Remember that in order to determine whether two Strings are equal you must use the `equals` method.

Assessment criteria

The marks will be divided as follows:

Implementation

Each of the methods will be assessed for correctness and for achieving the specified cost. Be sure to consider boundary cases. Unlike the Java module, we will not be assessing Java style. The marks allocated to each method are:

- longestWord: 25%
- countUnique: 25%

- findElement: 25%
- countGreaterOrEqual: 25%

Testing

You should carefully test your solutions, taking account of possible boundary cases. The distribution comes with four test files `tiny.txt`, `small.txt`, `medium.txt`, and `large.txt`. You should use these to test that your code is giving the correct answers. Below I have tabulated some of these, where `strings` is the StringList containing the data (note that your final solution will be tested with different data)

Method invocation	tiny	small	medium	large
longestWord(strings)	1	14	592	12473
countUnique(strings)	7	100	2293	62831
findElement(strings,"the")	5	85	2056	57384
findElement(strings,"ziggurat")	-1	-1	-1	-1
countGreaterOrEqual(strings,"its")	7	74	1261	34032
countGreaterOrEqual(strings,"orange")	5	50	941	25916

Submission

You are to submit a single file called `Search.java` as described above.

All submissions will be compiled and run by an automated process, so your class **must** build with the classes I have supplied. Do not submit versions of the other files; your class must compile with my originals.