Programming Assignment 1

CSC 225 Summer 2023

May 30, 2023

1 Creating Groups of Frogs

As per usual, you are figuring out how to group frogs. You must create a Java class called `Group.java` so that you are able to create groups of frogs. You may assume `Group.java` will only be used to group frogs (ie. do not make it generic).

Groups of frogs must always be ordered in alphabetical order by name. Frog names are case sensitive. Frog names are written entirely using the 26 letters of the English alphabet. The following methods must be implemented.

Method header	Description	Run-time efficiency*
`public void addFrog(Frog f)`	adds frog `f` to the group and places it in alphabetical order within the group	$O(\log n)$
`public Group[] halfGroups()`	returns a `Group[]` array with two elements, `g1` and `g2` in that order. `g1` contains the frogs in the group from indices 0 to $\lfloor \frac{n}{2} \rfloor - 1$ inclusive. `g2` contains all other frogs in the group	$O(n)$
`public Frog get(int i)`	returns the `Frog` at index `i`	$O(1)$
`public int size()`	returns the number of frogs in the group	$O(1)$
`public String toString()`	returns a String of the list of frogs that identically matches how `Arrays.toString(fs)` would print out a `Frog[]` array `fs`	$O(1)$

You may not use sorting of any kind. You must add each frog directly in its correct index (with respect to alphabetical order) each time a call to `addFrog()` is made.

*You **must** use `java.util.ArrayList`. Although it is unrealistic, we will assume `ArrayList` method calls to run in $O(1)$ time; that is, only your code's run-time efficiency will count.

The `Group` class must have exactly two attributes.

Data type	Description
`ArrayList<Frog>`	Contains the list of frogs currently in the group, organized alphabetically
`int`	The number of frogs currently in the group

2 Comparing Frog Groups

Comparing frog groups is an art. Two frog groups $G1$ and $G2$ are known to be "frog-equal" if they contain the same number of frogs and at least **one** of the following criteria is met:

a) $G1 = G2$, meaning the frogs in the corresponding indices are the same (have the same name)

If the number of frogs in each group is even, then we split the groups (from left to right) into two groups of the same size. $G1$ becomes $G1_{top}$, $G1_{bottom}$, and $G2$ becomes $G2_{top}$, and $G2_{bottom}$ in this manner. We can then check the following criteria:

b) $G1_{top}$ is frog-equal to $G2_{bottom}$

c) $G2_{top}$ is frog-equal to $G1_{bottom}$

Note: if $G1$ and $G2$ do not have an even number of frogs, then conditions b) and c) cannot be satisfied.

2.1 Examples

1. `G1 = [A,B]` and `G2 = [A]`

FALSE. `G1` and `G2` are not the same size.

2. `G1 = [A,B,C,D,E]` and `G2 = [A,B,C,D,E]`

TRUE. `G1` and `G2` have the same elements in the same indices.

3. `G1 = [A,B,C,H]` and `G2 = [H,I,J,K]`

TRUE. `G1` and `G2` are eventually split into four singleton groups and the two `H`'s are compared, resulting in a return value of `true`.

4. `G1 = [A,B,C,D,E,F]` and `G2 = [F,G,H,I,J,K]`

FALSE. `G1` is split into `G1_top = [A,B,C]` and `G1_bottom = [D,E,F]`. `G2` is split into `G2_top = [F,G,H]` and `G2_bottom = [I,J,K]`. Here, neither `G1_top` frog-equals `G2_bottom`, nor `G1_bottom` frog-equals `G2_top` since they contain an odd number of frogs and cannot be split further into two sub-groups.

2.2 Method description

Create a recursive method `FrogEquals()`. Note: it **must** be recursive in order to receive full marks.

Method header	Description	Run-time efficiency
`public static boolean FrogEquals(Group g1, Group g2)`	returns `true` if `g1` frog-equals `g2` and `false` otherwise	$O(n^2)$

3 Submission

You must submit all your work in Java. You must use the template provided `Group.java`. **Do not change the name or any of the provided code in this file.** Simply add your solutions to the existing code.

You are given two files: `Frog.java` and `Tester.java`. `Frog.java` contains the `Frog` class and its methods. When submitting your assignment, submit only `Group.java`.

4 Evaluation

The programming assignment will be marked out of 20, based on a combination of automated testing and human inspection.

Part	Max points
Part 1	10
Part 2	10

The following score ranges will apply to this assignment:

Score	Description
0-5	Submission does not compile
5-10	Compiles but mostly incorrect results
10-15	Mostly correct results but inefficient (does not follow run-time efficiency outlined)
15-20	Entirely correct results and efficient (follows run-time efficiency outlined)