Same Fish

Rar the Cat is at the local wet market to buy some fish. At a particular fish stall, there are **N** fishes placed on sale, arranged on a straight line from left to right. For convenience, the **K**-th fish from the left is labelled as fish **K**, where **K** is 1-indexed from 1 to **N**. Each fish is identified by its type, eg: Salmon, Tuna, etc… There can be multiple fishes of the same type.

Rar the Cat is not the only cat shopping for fishes. From time to time, a certain fish will be bought by other cats and the fishmonger will replace it with another fish, possibly of a **same or different type**. Rar the Cat wants to buy some amount of fishes **of the same type** to cook fish soup. However, he is lazy to walk throughout the whole fish stall and select fishes one by one. As such, he will **select a consecutive range of fishes** (consecutive fish labels) and intends to buy them.

Sadly, Rar the Cat cannot differentiate between the different fish types quickly. Hence, he employs you to determine whether the range of fishes he intends to purchase are of the same type or not. As Rar the Cat is fickle minded, there may be multiple such requests throughout the day, while other cats are purchasing their respective fishes. In total, there will be a total of **E** events. Each event can be one of the following:

- `REPLACE [K] [TYPE]` – The **K**-th fish from the left has been sold and is now replaced by another fish of type **[TYPE]**. <u>Please note that the replaced fish can be of the same type</u>.
- `SAME [L] [R]` – Rar the Cat asks you if the **L**-th fish to the **R**-th fish (inclusive) is of the same type. If it is, output the type. If not, output "DIFFERENT".

Input

The first line of input will contain an integer, **N**. This denotes the number of fishes that are on sale in the fish stall. The next **N** lines will be fish types, the **i**-th line that follows will be the fish type of the **i**-th fish from the left.

The next line of input will contain an integer **E**, the number of events. Each event will have one of the following input formats:

- `REPLACE [K] [TYPE]` – The **K**-th fish from the left has been sold and is now replaced by another fish of type **[TYPE]**.
- `SAME [L] [R]` – Rar the Cat asks you if the **L**-th fish to **R**-th fish (inclusive) are of the same type. If it is, output the type. If not, output "DIFFERENT".

Output

For every 'SAME' event, output the type of fish if the **L**-th fish to **R**-th fish are of the same type. If they are not of the same type, output "DIFFERENT" in capital letters.

Limits

- 0 < **N** ≤ 100,000, 0 < **E** ≤ 300,000.
- For 'replace' events: 0 < **K** ≤ **N**.
- For 'same' events: 0 < **L** ≤ **R** ≤ **N.**
- All fish types will contain only upper and lowercase alphabets with no spaces. (i.e. 'a' to 'z' and 'A' to 'Z'. Fish types are case sensitive, 'SALMON' and 'Salmon' are considered different fishes.

Grading

Your maximum correctness marks will be ***capped*** based on the program's time complexity as stated below. If your program's time complexity is not listed here, it will be assumed to be the next *slowest* time complexity.

- Slower than O(**NE**)– **10** out of 50
- O(**NE**) – **15** out of 50
- O(**(N+E)** sqrt **(N+E)**) – **30** out of 50
- O(**(N+E)** log **(N+E)**)– **50** out of 50 (includes O((N+E) log N) and O((N+E) log E))
  Hint: Consider the case where there is only 1 fish that is different from the rest. Please be reminded that you can use more than 1 data structures, including arrays.

Sample Testcase

Sample Input (**samefish1.in**)	Sample Output (**samefish1.out**)

`5`
`Tuna`
`Salmon`
`Catfish`
`Salmon`
`Salmon`
`5`
`SAME 3 5`
`SAME 4 5`
`REPLACE 3 Salmon`
`SAME 2 5`
`SAME 1 1`
	`DIFFERENT`
	`Salmon`
	`Salmon`
	`Tuna`

The fish stall displays a total of **N** = **5** fishes at any point of time. Initially, the fishes are [Tuna, Salmon, Catfish, Salmon, Salmon] from left to right. There are **E** = 5 events that follows:

1. Rar the Cat asks if fishes 3 to 5 are the same. This corresponds to [Catfish, Salmon, Salmon] and the result is that they are different.
2. Rar the Cat asks if fishes 4 to 5 are the same. This corresponds to [Salmon, Salmon] and the result is that they are the same – all are 'Salmon'.
3. Fish 3, originally 'Catfish' is now replaced with 'Salmon'. Now, the fishes are [Tuna, Salmon, Salmon, Salmon, Salmon] from left to right.
4. Rar the Cat asks if fishes 2 to 5 are the same. This corresponds to [Salmon, Salmon, Salmon, Salmon] and the result is that they are the same – all are 'Salmon'.
5. Rar the Cat asks if fishes 1 to 1 are the same. This corresponds to a single fish: Tuna.

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. Usage of methods in *Arrays* and *Collections* class such as *sort* and *binarySearch* are **allowed**. You are also allowed to use any linear data structures, non-linear data structures, arrays or strings such as LinkedList, ArrayList, HashSet, HashMap, TreeSet, TreeMap and PriorityQueue.
3. You are **free to define your own classes (or remove existing ones)** if it is suitable.
4. Please be reminded that the marking scheme is:

Input & Output	: 20% (10% each)
Correctness	: 50%
Programming Style	: 30% (awarded if you score **at least 20% from the above**):

   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

Compilation Error	: Deduction of **50% of the total marks obtained**.
Violation of Restrictions	: Deduction of up to **100% of the total marks obtained**.

Skeleton File – SameFish.java

You are given the skeleton file `SameFish.java`. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

You should see the following contents when you open the skeleton file:

```java
import java.util.*;
public class SameFish {
    private void run() {
        //implement your "main" method here
    }
    public static void main(String[] args) {
        SameFish newSameFish = new SameFish();
        newSameFish.run();
    }
}
```