Elections

The Presidential Elections in Pandaville has begun and there are N candidates. Mr. Panda has tasked you to help him code a program to help him collate the votes from the various cities in Pandaville. Each city has a voting station that sends in the votes into a voting collation system at the capital. To reduce the amount of traffic going into the capital, each voting station will collate some number of votes and send them in bulk instead of sending them one by one.

However, Pandaville has a unique voting system where citizens are allowed to downvote candidates on top of voting for them. Thus, the number of votes may be updated by a negative value. This also means the total number of votes for a candidate can be negative, but that candidate can still win the election if the candidate has the "highest" number of votes (-1 is higher than -2).

In total, there will be Q requests of the following form:

- `UPDATE [K] [CANDIDATE]` – Update K votes to candidate [CANDIDATE]

On top of the collation system, the citizens of Pandaville are even more demanding and they want a live update of who currently has the most votes and is in the running for president. Thus, after each update, you need to find out who currently has the most votes and if there is a tie, you need to show that as well.

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(NQ)– 10 out of 50
- O(NQ) – 15 out of 50
- O((N+Q) sqrt (N+Q)) – 30 out of 50
- O((N + Q) log N)– 50 out of 50
  Hint: Consider using more than 1 data structures in your algorithm.

Input

The first line of input will contain an integer, N. This denotes the number of candidates that are running for the election.

The next N lines will contain the number of a candidate, followed by a space and then the current number of votes that candidate has.

The next line of input will contain an integer Q, denoting the number of requests. Each request will have the following format:

- `UPDATE [K] [CANDIDATE]` – Update K votes to candidate [CANDIDATE]

Limits

- $0 < N \leq 100{,}000$, $0 < Q \leq 100{,}000$
- $-10{,}000 \leq K \leq 10{,}000$, $-10{,}000 \leq$ starting number of votes $\leq 10{,}000$
- All candidate names will contain only upper and lowercase alphabets with no spaces. (i.e. 'a' to 'z' and 'A' to 'Z'). There will not be a candidate called 'UPDATE'. Candidate names are case sensitive, 'PROFTAN' and 'Proftan' are considered different candidates.

Output

After every update, output a single line. If there are at least 2 people sharing the highest number of votes, output "Tie at **[number of votes]** votes!". Otherwise, output "**[candidate name]** is winning with **[number of votes]** votes!". Refer to the sample test case below for more information.

Sample Testcase

Sample Input (**elections1.in**)	Sample Output (**elections1.out**)
3 Proftan 5 Rarthecar -2 Rarthecat 5 6 UPDATE 4 Rarthecat UPDATE -20 Rarthecat UPDATE -7 Proftan UPDATE -9 Proftan UPDATE -9 Rarthecar UPDATE 10000 Rarthecar	Rarthecat is winning with 9 votes! Proftan is winning with 5 votes! Tie at -2 votes! Rarthecar is winning with -2 votes! Tie at -11 votes! Rarthecar is winning with 9989 votes!

Sample Explanation

	Proftan	Rarthecar	Rarthecat
Initial	5	-2	5
UPDATE 4 Rarthecat	5	-2	9
UPDATE -20 Rarthecat	5	-2	-11
UPDATE -7 Proftan	-2	-2	-11
UPDATE -9 Proftan	-11	-2	-11
UPDATE -9 Rarthecar	-11	-11	-11
UPDATE 10000 Rarthecar	-11	9989	-11

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 – Elections.java

You are given the skeleton file `Elections.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:

```
import java.util.*;

public class Elections {
    private void run() {
        //implement your "main" method here
    }
    public static void main(String[] args) {
        Elections newElections = new Elections();
        newElections.run();
    }
}

/**
 * A class to represent a candidate with a name and some number of votes
 * You may choose to use this to reduce the amount of code in your Elections class
 * However, you will not be penalised if you choose not to use it
 */
class Candidate implements Comparable {

    private String name;
    private int votes;

    public Candidate(String name, int votes) {
        this.name = name;
        this.votes = votes;
    }

    public String getName() {
        return name;
    }

    public int getVotes(){
        return votes;
    }

    @Override
    public int compareTo(Object other) {
        // implement this
    }
}
```