**Design Search Autocomplete System**

Design a search autocomplete system for a search engine. Users may input a sentence (at least one word and end with a special character '#').

You are given a string array sentences and an integer array times both of length n where sentences[i] is a previously typed sentence and times[i] is the corresponding number of times the sentence was typed. For each input character except '#', return the top 3 historical hot sentences that have the same prefix as the part of the sentence already typed.

Here are the specific rules:

* The hot degree for a sentence is defined as the number of times a user typed the exactly same sentence before.
* The returned top 3 hot sentences should be sorted by hot degree (The first is the hottest one). If several sentences have the same hot degree, use ASCII-code order (smaller one appears first).
* If less than 3 hot sentences exist, return as many as you can.
* When the input is a special character, it means the sentence ends, and in this case, you need to return an empty list.

Implement the AutocompleteSystem class:

* AutocompleteSystem(String[] sentences, int[] times) Initializes the object with the sentences and times arrays.
* List<String> input(char c) This indicates that the user typed the character c.
  + Returns an empty array [] if c == '#' and stores the inputted sentence in the system.
  + Returns the top 3 historical hot sentences that have the same prefix as the part of the sentence already typed. If there are fewer than 3 matches, return them all.

**Example 1:**

**Input**

["AutocompleteSystem", "input", "input", "input", "input"]

[[["i love you", "island", "iroman", "i love leetcode"], [5, 3, 2, 2]], ["i"], [" "], ["a"], ["#"]]

**Output**

[null, ["i love you", "island", "i love leetcode"], ["i love you", "i love leetcode"], [], []]

**Explanation**

AutocompleteSystem obj = new AutocompleteSystem(["i love you", "island", "iroman", "i love leetcode"], [5, 3, 2, 2]);

obj.input("i"); // return ["i love you", "island", "i love leetcode"]. There are four sentences that have prefix "i". Among them, "ironman" and "i love leetcode" have same hot degree. Since ' ' has ASCII code 32 and 'r' has ASCII code 114, "i love leetcode" should be in front of "ironman". Also we only need to output top 3 hot sentences, so "ironman" will be ignored.

obj.input(" "); // return ["i love you", "i love leetcode"]. There are only two sentences that have prefix "i ".

obj.input("a"); // return []. There are no sentences that have prefix "i a".

obj.input("#"); // return []. The user finished the input, the sentence "i a" should be saved as a historical sentence in system. And the following input will be counted as a new search.

**Constraints:**

* n == sentences.length
* n == times.length
* 1 <= n <= 100
* 1 <= sentences[i].length <= 100
* 1 <= times[i] <= 50
* c is a lowercase English letter, a hash '#', or space ' '.
* Each tested sentence will be a sequence of characters c that end with the character '#'.
* Each tested sentence will have a length in the range [1, 200].
* The words in each input sentence are separated by single spaces.
* At most 5000 calls will be made to input.

/\*\*

\* @param {string[]} sentences

\* @param {number[]} times

\*/

var AutocompleteSystem = function(sentences, times) {

};

/\*\*

\* @param {character} c

\* @return {string[]}

\*/

AutocompleteSystem.prototype.input = function(c) {

};

/\*\*

\* Your AutocompleteSystem object will be instantiated and called as such:

\* var obj = new AutocompleteSystem(sentences, times)

\* var param\_1 = obj.input(c)

\*/