H1

Word Count Engine

- Pramp Link
- HashMaps
- 2019, Google, Pramp

H2

Problem

Implement a document scanning function `wordCountEngine`, which receives a string `document` and returns a list of all unique words in it and their number of occurrences, sorted by the number of occurrences in a descending order. If two or more words have the same count, they should be sorted according to their order in the original sentence. Assume that all letters are in english alphabet. You function should be case-insensitive, so for instance, the words `“Perfect”` and `“perfect”` should be considered the same word.

The engine should strip out punctuation (even in the middle of a word) and use whitespaces to separate words.

Analyze the time and space complexities of your solution. Try to optimize for time while keeping a polynomial space complexity.

Examples:

```
input:  document = "Practice makes perfect. you'll only
                    get Perfect by practice. just practice!"

output: [ ["practice", "3"], ["perfect", "2"],
          ["makes", "1"], ["youll", "1"], ["only", "1"],
          ["get", "1"], ["by", "1"], ["just", "1"] ]
```

Important: please convert the occurrence integers in the output list to strings (e.g. `"3"` instead of `3`). We ask this because in compiled languages such as C#, Java, C++, C etc., it’s not straightforward to create mixed-type arrays (as it is, for instance, in scripted languages like JavaScript, Python, Ruby etc.). The expected output will simply be an array of string arrays.

Constraints:

- **[time limit] 5000ms**
- **[input] string** `document`
- **[output] array.array.string**

H2

Solution

The Problem can be divided into 3 categories

Category 01

- Parse the Input into a String array
- Ensure similar words are all lowercase, trimp Space, remove special characters ..etc.,

Category 02

H3

- Used a Hashmap to store the unique words
- HashMap has key (words) and value (number of occurances)
- Iterate through the input array & Increase the number of occurance if the word matches

H3

Category 03

- Convert the HashMap to the suitable array
- Ensure the array is sorted by the largest occurance first *(not done)*

H3

```
// let document = "Practice makes perfect. you'll only get Perfect by practice. just practice!";
// document = "Practice makes perfect. you'll only get Perfect. practice practice";
// document = "practice practice";
let document = "Practice makes perfect, you'll get perfecT by practice. just practice! just just just!!";

// Category 01
const wordToArray = (input) => {
    return input.split(' ').map(item => item.toLowerCase().replace(/[&\/\\#,+!()$~%.":*?<>{}]/g, '')).sort();
};

// Category 02
const wordCountMap = (arr) => {
    const hasWords = new Map();
    arr.map(element => {
        if (hasWords.has(element)){
            hasWords.set(element, hasWords.get(element) + 1);
        } else {
            hasWords.set(element, 1);
        }
    });
    return hasWords;
};
```

```
// Category 03
const mapToList = (map) => {
    let result = [];
    for (let [key, value] of map.entries()) {
        result.push([key , value]);
      }
    return result;
}


const result = wordToArray(document); // category 01
const resultMap = wordCountMap(result); // category 02
console.log(mapToList(resultMap)); // category 03
```

H2

Time Space Complexity

Each of the category increments through the Array **atleast** one time

- Category 01 -> O(n) + O(n) + O(n) = O3(n) ~ O(N)
- Category 02 -> O(n) ~ O(N)
- Category 03 -> O(n) ~ O(N)

As the size of the array increases, it still has to go through all three categories ie, O(3N) ~ **O(N)** times