Group Anagrams - Notes

Problem Statement

Given an array of strings `strs`, group the anagrams together. An anagram is a word formed by rearranging the letters of another word.

Approach

1. **Sort Each Word:** Convert each string into a character array, sort it, and convert it back to a string. This sorted string will be used as a key in a HashMap.
2. **Use HashMap to Group Anagrams:**
   - If the sorted string is already in the map, add the original word to its corresponding list.
   - If not, create a new list and store the word.
3. **Return all groups as a list of lists.**

Code Implementation (Java)

```
import java.util.*;

class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        HashMap<String, List<String>> big = new HashMap<>();

        for (String word : strs) {
            char[] chars = word.toCharArray();
            Arrays.sort(chars);
            String sortedStr = new String(chars);

            big.putIfAbsent(sortedStr, new ArrayList<>()); // Create a new list if
key doesn't exist
            big.get(sortedStr).add(word); // Add word to the correct group
        }

        return new ArrayList<>(big.values()); // Convert values to a list
```

```
    }
}
```

Key Concepts

1. **Sorting a String**

```
char[] chars = word.toCharArray();
Arrays.sort(chars);
String sortedStr = new String(chars);
```

   - Converts the string into a char array
   - Sorts the array
   - Converts it back into a string

2. **Using HashMap for Grouping**

```
big.putIfAbsent(sortedStr, new ArrayList<>());
big.get(sortedStr).add(word);
```

   - If `sortedStr` is not in `big`, create a new list.
   - Add the word to its respective list.

3. **Returning the Values as List of Lists**

```
return new ArrayList<>(big.values());
```

   - `big.values()` gives a collection of lists.
   - `new ArrayList<>(big.values())` converts it into a list of lists.

Time Complexity Analysis

- **Sorting each string:** `O(k log k)`, where `k` is the length of the longest string.
- **Processing `n` strings:** `O(n * k log k)`.
- **HashMap Operations:** `O(1)` on average.
- **Overall Complexity:** `O(n * k log k)`.

Mistakes I Made

1. **Didn't sort the string before using it as a key.**
2. **Used `HashMap<String, String>` instead of `HashMap<String, List<String>>`.**
3. **Forgot to check if key exists before adding to the list.**

Final Thoughts

- Sorting is the key idea in this problem.
- HashMap helps in grouping anagrams efficiently.
- Make sure to use `putIfAbsent` to handle new groups.