🧠 LeetCode 49 — Group Anagrams

📌 Problem Summary

Given an array of strings `strs`, we need to **group words that are anagrams** of each other.

> Anagram → Two strings that contain the same characters with the same frequency but possibly in different order.

✅ Example

Input:

```
["eat","tea","tan","ate","nat","bat"]
```

Output:

```
[["bat"],["nat","tan"],["ate","eat","tea"]]
```

🎯 Key Idea

If two words are anagrams, then when we **sort their characters**, both will become the same string.

Example:

Word	Sorted
eat	aet
tea	aet
tan	ant

So we use a **hash map**:

- Key → sorted word

- Value → list of words that match this sorted key

💡 Approach

1. Iterate through each string
2. Sort the string (this acts as its signature)
3. Store original string in map under the sorted key
4. Return values of the map

🧾 Code (C++)

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& arr) {
        vector<vector<string>> ans;
        int n = arr.size();
        unordered_map<string,vector<string>> mp;
        for(int i=0; i<n; i++){
            string lexo = arr[i];
            sort(lexo.begin(),lexo.end());
            if(mp.find(lexo) == mp.end()){
                mp[lexo] = {arr[i]};
            }
            else{
                mp[lexo].push_back(arr[i]);
            }
        }
        for(auto ele : mp){
            ans.push_back(ele.second);
        }
        return ans;
    }
};
```

🕒 Time & Space Complexity

Complexity	Value
Time	**O(N * K log K)** (N = words, K = max length of a word)
Space	**O(N * K)**

✨ Takeaways

- Sorted string acts like a **fingerprint** for anagrams
- Hash map helps in **efficient grouping**
- Classic use of **string manipulation + hashing**
