Multiple Anagrams

Q: Given an array of strings, group the anagrams together. You can return the answers in any order

Clarifications:
- How should I handle duplicate strings?
- Definition of an anagram
- Is each string not empty?

Test Cases:
1. ["bat", "eat", "nab", "ate", "fight", "tea"]
   ↳ [["bat", "nab"], ["eat", "tea", "ate"], ["fight"]]
2. ["abc", "def", "ghi"]
   ↳ [["abc"], ["def"], ["ghi"]]
3. ["abc", "bca", "cba"]
   ↳ [["abc", "bca", "cba"]]

Algorithm:
1. Compare number of characters
   ↳ if not equal, not anagrams
2. Check if strings are identical
   ↳ if so, they are anagrams
3. Add first string to hashmap by letter and frequency
4. Iterate through each character of each subsequent string
5. If there is a match, remove from list and add original string + anagram to 2D Array
   ↳ Once end of list is reached, remove og string as well
6. If no match, just add to 2D array

Multiple Anagrams (continued)

strs[][] *

[illegible]

```
def multipleAnagrams(self, strs):
    grouped = [][]

    for i in range len(strs):
        for j in range(i, len(strs)):
            if strs[i] == strs[j]:
                grouped[i][j] =
```

Neetcode Notes:

- Any two strings are anagrams of each other if when sorted, they are equivalent
  - ↳ to do this for each string is a time complexity of $O(m \cdot n \log n)$ where n is the average length and m is the total # of strings
- However, each string can only have chars from a-z (26 unique chars)
- You can make a hashmap that counts how many of each character each string has, using that as the key, and the value is the list of anagrams
- Time Complexity: $O(m \cdot n \cdot 26) = O(m \cdot n)$
  - Make a dictionary that takes an array of character counts and maps that to a list of anagrams

Multiple Anagrams (Neetcode)

```
def groupAnagrams (self, strs):

        # mapping character count to a list of anagrams
        res = defaultdict(list)
        for s in strs: # iterate through each string
            count = [0] * 26 # zero-index, a-z

            for c in s:
                count[ord(c) - ord("a")] += 1
                # the above line subtracts each char from the
                # ASCII value of lowercase a, such that a
                # is index 0, b is 1.... so on and so forth

            res[tuple(count)].append(s)

        return res.values()
        # Time Complexity of O(m·n)
```

Analysis Algorithmically:

- Create an empty hash map where keys represent the character counts and values are the words that match
- Loop through each string
- Create and init a list of 26 zeroes (for each alpha letter)
- Loop through each char in current word
- Assign each character to the appropriate spot in the array
- Cast the letter list as a tuple to use it as a dict key
- Add the original word to the dictionary
- Return the values