Our Solution(s) Run Code

Solution 1 Solution 2

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

import (
  "sort"
)

// O(w * n * log(n) + n * w * log(w)) time | O(wn) space - where
// n is the length of the longest word
func GroupAnagrams(words []string) [][]string {
  if len(words) == 0 {
    return [][]string{}
  }

  sortedWords := []string{}
  indices := []int{}
  for i, word := range words {
    sortedWords = append(sortedWords, sortWord(word))
    indices = append(indices, i)
  }
  sort.Slice(indices, func(i, j int) bool {
    return sortedWords[indices[i]] < sortedWords[indices[j]]
  })

  result := [][]string{}
  currentAnagramGroup := []string{}
  currentAnagram := sortedWords[indices[0]]
  for _, index := range indices {
    word := words[index]
    sortedWord := sortedWords[index]
    if len(currentAnagramGroup) == 0 {
      currentAnagramGroup = append(currentAnagramGroup, word)
```

Your Solutions Run Code

Solution 1 Solution 2 Solution 3

```go
package main

func GroupAnagrams(words []string) [][]string {
  // Write your code here.
  return nil
}

```

Our Tests Custom Output Submit Code

```
import (
    "sort"


    "github.com/stretchr/testify/require"
)

func (s *TestSuite) TestCase1() *TestCase {
    words := []string{}
    expected := [][]string{}
    output := GroupAnagrams(words)
    compare(t, expected, output)
}

func (s *TestSuite) TestCase2() *TestCase {
    words := []string{"test"}
    expected := [][]string{{"test"}}
    output := GroupAnagrams(words)
    compare(t, expected, output)
}
```

Run or submit code when you're ready.