Problem 1:

Given an array of strings, group anagrams together.

Example:

- **Input**: ["eat", "tea", "tan", "ate", "nat", "bat"]
- **Output**: [["eat", "tea", "ate"], ["tan", "nat"], ["bat"]]

Additional Examples:

Example 1:

- **Input**: [""]
- **Output**: [[""]] (single empty string)

Example 2:

- **Input**: ["a"]
- **Output**: [["a"]] (single character string)

Example 3:

- **Input**: ["abc", "bca", "cab", "xyz", "zyx", "yxz"]
- **Output**: [["abc", "bca", "cab"], ["xyz", "zyx", "yxz"]]

Example 4:

- **Input**: ["listen", "silent", "enlist", "inlets", "google", "gogole"]
- **Output**: [["listen", "silent", "enlist", "inlets"], ["google", "gogole"]]

Example 5:

- **Input**: ["rat", "tar", "art", "car", "arc"]
- **Output**: [["rat", "tar", "art"], ["car", "arc"]]

Problem 2:

Given a string *ss*, return the longest palindromic substring in *ss*.

Example 1:

- **Input**: s = "babad"
- **Output**: "bab" (or "aba")

Example 2:

- **Input**: s = "cbbd"
- **Output**: "bb"

Additional Examples:

Example 3:

- **Input**: s = "a"
- **Output**: "a" (single character string)

Example 4:

- **Input**: s = "ac"
- **Output**: "a" (or "c")

Example 5:

- **Input**: s = "racecar"
- **Output**: "racecar" (the entire string is a palindrome)

Example 6:

- **Input**: s = "forgeeksskeegfor"
- **Output**: "geeksskeeg"

Problem 3:

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

A string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

- **Input**: s = "()[]{}"
- **Output**: true

Example 2:

- **Input**: s = "(]"
- **Output**: false

Additional Examples:

Example 3:

- **Input**: s = "([{}])"
- **Output**: true

Example 4:

- **Input**: s = "([)]"
- **Output**: false

Example 5:

- **Input**: s = "{[]}"
- **Output**: true

Example 6:

- **Input**: s = "((()))"
- **Output**: true

Problem 4:

Given two strings *ss* and *tt*, return the smallest substring in *ss* that contains all the characters in *tt*. If no such substring exists, return an empty string "".

Example 1:

- **Input**: s = "ADOBECODEBANC", t = "ABC"
- **Output**: "BANC"

Example 2:

- **Input**: s = "a", t = "a"
- **Output**: "a"

Example 3:

- **Input**: s = "a", t = "aa"
- **Output**: "" (no such substring exists)

Additional Examples:

Example 4:

- **Input**: s = "this is a test string", t = "tist"
- **Output**: "t stri"

Example 5:

- **Input**: s = "aaflslflsldkalskaaa", t = "aaa"
- **Output**: "aaa"

Problem 5:

Two strings are considered close if you can swap letters or change the frequency of any letter to match the other string. Determine if two given strings are close.

Example 1:

- **Input**: word1 = "abc", word2 = "bca"
- **Output**: true

Example 2:

- **Input**: word1 = "aabbcc", word2 = "xxyyzz"
- **Output**: false

Additional Examples:

Example 3:

- **Input**: word1 = "abbzzca", word2 = "babzzcz"
- **Output**: true

Example 4:

- **Input**: word1 = "aabbccdd", word2 = "ddccbbaa"
- **Output**: true

Example 5:

- **Input**: word1 = "abc", word2 = "def"
- **Output**: false

Example 6:

- **Input**: word1 = "a", word2 = "aa"
- **Output**: false

Problem 6:

Given two strings *ss* and *tt*, determine if *tt* is an anagram of a substring of *ss*. In other words, check if there exists a substring in *ss* that is an anagram of *tt*.

Example 1:

- **Input**: s = "cbaebabacd", t = "abc"
- **Output**: true (The substring "cba" is an anagram of "abc")

Example 2:

- **Input**: s = "af", t = "be"

- **Output**: false (No substring of "af" is an anagram of "be")

Additional Examples:

Example 3:

- **Input**: s = "abacbabc", t = "abc"
- **Output**: true (The substring "bac" is an anagram of "abc")

Example 4:

- **Input**: s = "abcdefg", t = "gfedcba"
- **Output**: true (The entire string "abcdefg" is an anagram of "gfedcba")

Example 5:

- **Input**: s = "hello", t = "oll"
- **Output**: true(The substring of "llo" is an anagram of "oll")

Problem 7

Write a function to find the longest common prefix string amongst an array of strings.
If there is no common prefix, return an empty string "".

Example 1:
Input: strs = ["flower","flow","flight"]
Output: "fl"
Example 2:
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings

Problem 8

Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:
Input: haystack = "sadbutsad", needle = "sad"
Output: 0

Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
Example 2:
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.

Problem 9

Given a string s consisting of words and spaces, return *the length of the **last** word in the string.*
A **word** is a maximal
substring
consisting of non-space characters only.

Example 1:
Input: s = "Hello World"
Output: 5
Explanation: The last word is "World" with length 5.
Example 2:
Input: s = " fly me to the moon "
Output: 4
Explanation: The last word is "moon" with length 4.
Example 3:
Input: s = "luffy is still joyboy"
Output: 6
Explanation: The last word is "joyboy" with length 6.

Problem 10

Given a string paragraph and a string array of the banned words banned, return *the most frequent word that is not banned*. It is **guaranteed** there is **at least one word** that is not banned, and that the answer is **unique**.
The words in paragraph are **case-insensitive** and the answer should be returned in **lowercase**.

Example 1:
Input: paragraph = "Bob hit a ball, the hit BALL flew far after it was hit.", banned = ["hit"]
Output: "ball"

Explanation:
"hit" occurs 3 times, but it is a banned word.
"ball" occurs twice (and no other word does), so it is the most frequent non-banned word in the paragraph.
Note that words in the paragraph are not case sensitive,
that punctuation is ignored (even if adjacent to words, such as "ball,"),
and that "hit" isn't the answer even though it occurs more because it is banned.
Example 2:
Input: paragraph = "a.", banned = []
Output: "a"