OA2 Review

Question 1

1) 背景

Given a string s, find the longest palindromic substring in s.

2) My Soultion

The idea is to generate all even length and odd length palindromes and keep track of the longest palindrome seen so far.

Step 1. The first step is to check the edge case. If the length of the string is smaller or equal to 1, the function returns the string directly.

Step2. I set the current longest palindrome to be the first character. Then, I wrote a for-loop to set all possible palindromes centers to check all even and odd length palindromes. In each for-loop, there is a helper function to check the length of palindromes when setting center point is I or (i and I + 1).

Step3. In the help function, there are two pointers as a left pointer and right pointer to expand two sides. When the pointers cross the boundary or left value not equal to the right value, it will break the while loop. Finally, return the palindrome.

Step4. After generating all even length and odd length palindromes to update the longest palindrome, the function returns the longest palindrome.

3) Time complexity and spce complexity

Time complexity: $O(N^2)$ (quadratic)
Space complexity: $O(1)$ (linear)

4) Code

```
class Solution(object):
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        n = len(s)
        if n <= 1:
            return s
        maxLen = 1
        res = s[0:1]
        for i in range(n):
            tmp = self.helper(s, i, i)
            if len(tmp) > maxLen:
                res = tmp
                maxLen = len(tmp)

            tmp = self.helper(s, i, i + 1)
            if len(tmp) > maxLen:
                res = tmp
                maxLen = len(tmp)
        return res

    def helper(self, s, l, r):
        n = len(s)
        while(l >= 0 and r < n and s[l] == s[r]):
            l -= 1
            r += 1
        return s[l + 1:r]
```

2. Other solution

Another solution is to use dynamic programming. The idea is to main a boolean matrix dp[n][n] that filled in a bottom-up manner.
If the value of dp[i][j] is True, which means the substring str[i:j] is palindrome. If the value of dp[i][j] is False, which means the substring str[i:j] is not palindrome. We can find the longest palindrome by updating the dp matrix process.

2.1 Pesudo Code

Step 1: The first step is to check the edge case. If the length of the string is smaller or equal to 1, the function returns the string directly.

Step 2: Initialize the dp matrix

$$\text{dp[i][i] = True} \ \ 0 \le i < n$$

dp[i][i+1] = True if s[i] == s[i+1] else False $0 \leq i < n-1$

Step 3: Recursion Function

dp[i][j] = True if s[j] == s[i] and dp[i+1][j-1]

Step 4: Return the longest palindromic substrings.

2.2 Time Complexity and Space Complexity

Time Complexity: $O(N^2)$
Space Complexity: $O(N^2)$

2.3 Code

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        ans = ''
        if not s:
            return ans
        n = len(s)
        dp = [[0]*n for _ in range(n)]
        max_len = 1

        for i in range(n):
            dp[i][i] = True
            ans = s[i]

        for i in range(n-1):
            if s[i]==s[i+1]:
                dp[i][i+1] = True
                max_len = True
                ans = s[i:i+2]

        for j in range(1,n):
            for i in range(0, j-1):
                if s[j] == s[i] and dp[i+1][j-1]:
                    dp[i][j] = True
                    if j-i+1 > max_len:
                        max_len = j-i+1
```

```
                            ans = s[i:j+1]
            return ans
```

Question 2

Find the most common words in a sentence but not in a banned list.

Step1: Changed sentence to lowercase and then split it to list by each word.
Step2: Lower the banned list each word. At first, I didn't do this which led two test case cannot pass.
Step3: Initialize a dictionary to use work as key and number occurrence as value, a maxNum variable to update the largest number.
Step 4. Loop through the sentence list and update the dictionary and variable.
Step 5. Loop through the dictionary, if the value is equal to maxNum, then add the key to the return list.

1. Code

```
import re
class Solution(object):
    def mostCommonWord(self, paragraph, banned):
        """
        :type paragraph: str
        :type banned: List[str]
        :rtype: str
        """
        dict_word_num = {}
        res = []
        if not paragraph:
            return res
        maxNum = 1
        # paragraph = re.findall(r'\w+', paragraph.lower())
        paragraph = re.findall(r'[a-zA-Z]+', paragraph.lower())
        for word in paragraph:
            word = word.lower()
            if word in banned:
                continue
            if word in dict_word_num:
                dict_word_num[word] += 1
                if dict_word_num[word] > maxNum:
                    maxNum = dict_word_num[word]
```

```python
                else:
                    dict_word_num[word] = 1

        for word, num in dict_word_num.items():
            if num == maxNum:
                res.append(word)
        return res
```