212. Word Search II

We will use trie to store all words. Then iterate through all cells in board. If there is a letter in trie.root.children. We will use dfs to find the next letter. When we get to point that isWord is True, we will append the word to result and set isWord = False, so that we won't add words repeatedly into result. TC is O(n * length)

```
class TrieNode:
  def __init__(self):
    self.children = collections.defaultdict(TrieNode)
    self.isWord = False

class Trie:
  def __init__(self):
    self.root = TrieNode()

  def insert(self, word):
    node = self.root
    for w in word:
      node = node.children[w]
    node.isWord = True

class Solution:
    def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
        rows, cols = len(board), len(board[0])
        result = []
        trie = Trie()
        for word in words:
          trie.insert(word)
        for i in range(rows):
          for j in range(cols):
            if board[i][j] in trie.root.children:
              self.dfs(trie.root.children[board[i][j]], set([(i, j)]), board[i][j], i, j, board, rows, cols, result)
        return result

    def dfs(self, node, visited, cur, i, j, board, rows, cols, result):
      if node.isWord == True:
        result.append(cur)
        node.isWord = False

      for d_i, d_j in [[0, 1], [0, -1], [-1, 0], [1, 0]]:
        new_i, new_j = i + d_i, j + d_j
        if 0 <= new_i < rows and 0 <= new_j < cols and (new_i, new_j) not in visited and
board[new_i][new_j] in node.children:
          visited.add((new_i, new_j))
```

```
        self.dfs(node.children[board[new_i][new_j]], visited, cur + board[new_i][new_j], new_i,
new_j, board, rows, cols, result)
        visited.remove((new_i, new_j))
```

559. Maximum Depth of N-ary Tree

We will use recursion to get max depth of all of its children and return max(result) + 1. When node.children is empty, we will return 1, we node is None, we will return 0. TC is O(n)

```
class Solution:
    def maxDepth(self, root: 'Node') -> int:
        if not root:
          return 0
        if not root.children:
          return 1
        max_depth = 0
        for node in root.children:
           max_depth = max(max_depth, self.maxDepth(node))
        return max_depth + 1
```

500. Keyboard Row

We will use a hashmap to record each letter with the same value if they are on the same row. Then we will compare each word's adjacent letter. If they don't have equal value in memo, we will break. If we go through all letters and they are all the same, we will append this word to result. TC is O(n * length)

```
class Solution:
    def findWords(self, words: List[str]) -> List[str]:
        a = ['qwertyuiop','asdfghjkl','zxcvbnm']
        memo = {}
        result = []

        for i, letters in enumerate(a):
          for l in letters:
            memo[l] = i

        for word in words:
          mark = True
          for i in range(1, len(word)):
            if memo[word[i].lower()] != memo[word[i - 1].lower()]:
              mark = False
              break
          if mark:
            result.append(word)
        return result
```

476. Number Complement

It's a math question, we will get num's binary and get its bits number, then the result would be 2 ** (len(bits)) - 1. TC is O(1)

```
class Solution:
    def findComplement(self, num: int) -> int:
        binary = bin(num)[2:]
        return 2 ** (len(binary)) - 1 - num
```

1099. Two Sum Less Than K

We will sort the array and iterate from two sides and once two sum is larger or equal to K, we will deduct right, or add left and update our maximum sum. TC is O(nlogn)

```
class Solution:
    def twoSumLessThanK(self, A: List[int], K: int) -> int:
        if len(A) < 2:
          return -1
        A.sort()
        if A[0] + A[1] >= K:
          return -1

        left, right = 0, len(A) - 1
        max_sum = 0
        while left < right:
          if A[left] + A[right] >= K:
            right -= 1
          else:
            max_sum = max(max_sum, A[left] + A[right])
            left += 1
        return max_sum
```