301. Remove Invalid Parentheses

For this question, we could remove in two parts, the first one is to remove the redundant ‘)’, the second one is to remove ‘)’ from the previous result. When we remove ‘)’, it’s important to prevent the duplication. So we will remember the last index we remove and won’t remove the consecutive ‘(’s. The we will do it again for the reversed result to reuse the previous code. The TC is O(n*n):

```
class Solution(object):
    def removeInvalidParentheses(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        result = []
        self.removeHelper(s, result, 0, 0, ['(',')'])
        return result

    def removeHelper(self, s, result, last_i, last_j, par):
        stack = 0
        for i in range(last_i, len(s)):
            if s[i] == par[0]:
                stack += 1
            elif s[i] == par[1]:
                stack -= 1
            if stack < 0:
                for j in range(last_j, i + 1):
                    if s[j] == par[1] and (j == last_j or s[j - 1] != s[j]):
                        self.removeHelper(s[:j] + s[j + 1:], result, i, j, par)
                return

        reversed_s = s[::-1]
        if par[0] == '(':
            self.removeHelper(reversed_s, result, 0, 0, [')', '('])
        else:
            result.append(reversed_s)
```

121. Best Time to Buy and Sell Stock

It’s very simple, we only need to iterate through the whole array, use a variable to store minimum value of all elements. If the current value is larger than min_price, we would compare the current benefit with the difference of cur_price - min_price, and store the larger one to benefit. In the end, we could be able to get what we want.

```
 class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        min_num = float('inf')
```

```
        result = 0
        for price in prices:
            if min_num > price:
                min_num = price
            else:
                result = max(result, price - min_num)
        return result
```

937. Reorder Log Files
This question is quite simple. We only need to separate the original one into letter array and digit array and sort letter array by key=(str_without_id, str) to get ordered letter array. In the end, we will return letter + digit array. The TC is O(nlogn)

```
class Solution:
    def reorderLogFiles(self, logs: List[str]) -> List[str]:
        letters = []
        digits = []
        digits_map = '1234567890'
        for log in logs:
            if log.split(' ')[1][0] in digits_map:
                digits.append(log)
            else:
                letters.append(log)
        letters.sort(key=lambda a: (a[a.index(' ') + 1:], a))
        return letters + digits
```

76. Minimum Window Substring
For this question, we would use slide window. We will start from increasing right side of window to incorporate all characters in t, then move left side to right until the slide window cannot be able to include all characters in t. At that time, we will compare and get minimum length of string. Then repeat the above process until the right get out of right bound of s.

```
from collections import defaultdict
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        memo = defaultdict(int)
        for i in t:
            memo[i] += 1
        count = len(memo.keys())
        count_memo = count
        min_length = float('inf')
        min_str = ""
        left, right = 0, 0
```

```
        while right < len(s):
            while count > 0 and right < len(s):
                if s[right] in memo:
                    memo[s[right]] -= 1
                    if memo[s[right]] == 0:
                        count -= 1
                right += 1
            if count == 0:
                while left <= right - 1 and count == 0:
                    if s[left] in memo:
                        memo[s[left]] += 1
                        if memo[s[left]] == 1:
                            count += 1
                    left += 1
                if right - left + 1 < min_length:
                    min_length = right - left + 1
                    min_str = s[left - 1:right]
        return min_str
```

289. Game of Life

This question is very easy. We only need to figure out how to represent its next generation's state without removing this generation's state. So we could set decimal digit to represent neighbor's number. Then in the second traversal, we could replace its original with real state.

```
class Solution:
    def gameOfLife(self, board: List[List[int]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        if not board or not board[0]:
            return board
        directions = [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, 0], [-1, -1], [-1, 1]]
        rows = len(board)
        cols = len(board[0])
        for i in range(rows):
            for j in range(cols):
                count = 0
                for d_i, d_j in directions:
                    new_i = i + d_i
                    new_j = j + d_j
```

```
            if 0 <= new_i < rows and 0 <= new_j < cols and board[new_i][new_j] % 10 == 1:
                count += 1
        board[i][j] += count * 10

for i in range(rows):
    for j in range(cols):
        result = board[i][j] // 10
        if result < 2 or result > 3:
            board[i][j] = 0
        elif result == 3:
            board[i][j] = 1
        else:
            board[i][j] = board[i][j] % 10
return board
```