560. Subarray Sum Equals K
This question is quite tricky, for sum[i, j], we could get from sum[0,j] - sum[0,i], we could store sum[0, i] using map<sum, times>, so in the next time, we could check whether current sum - k exists in the map, if it does, we could accurate result by associated times. TC would be O(n)

```
from collections import defaultdict
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        cur_sum = 0
        sum_memo = defaultdict(int)
        sum_memo[0] = 1
        result = 0
        for num in nums:
            cur_sum += num
            if cur_sum - k in sum_memo:
                result += sum_memo[cur_sum - k]
            sum_memo[cur_sum] += 1
        return result
```

771. Jewels and Stones
This question is quite simple. We will transform J to set and iterate through S and calculate the number of jewel in S. The TC is O(n)

```
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        jewels = set(list(J))
        result = 0
        for s in list(S):
            if s in jewels:
                result += 1
        return result
```

22. Generate Parentheses
For this question, we could use bfs or dfs. I choose to use bfs because it could prevent stack overflow. We don’t need recursion. The TC is O(n * n):

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        next_ite = []
        cur_ite = [('(', 1, 0)]
        for i in range(n * 2 - 1):
            for s, l, r in cur_ite:
                if l - r > 0:
```

```
                next_ite.append((s + ')', l, r + 1))
            if l < n:
                next_ite.append((s + '(', l + 1, r))
        cur_ite = next_ite
        next_ite = []
    return map(lambda a: a[0], cur_ite)
```

79. Word Search

For this question, we would use DFS, we will iterate through all cells in the board, once that cell's value equals to our word[0], we will find its cells in for directions to check the next character exists. Repeat this process until we reach all characters of our target word. Then we will return True. After finishing all these search, we will return False.

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        directions = [[0, 1], [0, -1], [1, 0], [-1, 0]]
        if not board or not board[0]:
            return False
        if not word:
            return True
        def findNextWord(index, visited, i, j):
            if index == len(word):
                return True
            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 (new_i, new_j) not in visited and
word[index] == board[new_i][new_j]:
                    visited.add((new_i, new_j))
                    if findNextWord(index + 1, visited, new_i, new_j):
                        return True
                    visited.remove((new_i, new_j))
            return False
        rows = len(board)
        cols = len(board[0])
        for i in range(rows):
            for j in range(cols):
                if board[i][j] == word[0]:
                    if findNextWord(1, set([(i, j)]), i, j):
                        return True
        return False
```

380. Insert Delete GetRandom O(1)

This question is quite tricky, because we need to implement getRandom in O(1), so we definitely an array to achieve this. But then we also need to implement insert and remove in O(1). We need a map to memorize each element's index so that it would be possible. When we remove some element, we will put the last element on that position and remove the last element of the array.

```
class RandomizedSet:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.nums = []
        self.pos = {}

    def insert(self, val: int) -> bool:
        """
        Inserts a value to the set. Returns true if the set did not already contain the specified element.
        """
        if val in self.pos:
            return False
        self.nums.append(val)
        self.pos[val] = len(self.nums) - 1
        return True

    def remove(self, val: int) -> bool:
        """
        Removes a value from the set. Returns true if the set contained the specified element.
        """
        if val in self.pos:
            index = self.pos[val]
            self.nums[index] = self.nums[-1]
            self.pos[self.nums[-1]] = index
            self.nums.pop()
            del self.pos[val]
            return True
        else:
            return False

    def getRandom(self) -> int:
        """
        Get a random element from the set.
        """
```

```
return self.nums[random.choice(range(len(self.nums)))]
```