692. Kth frequent words
We should use dict to record each word's occuring time. Then we will use heapq to push (-value, key) to a heap. Then we heappop k elements, which are what we want. TC is O(max(n, klogn))

```
from collections import Counter
from heapq import *
class Solution:
    def topKFrequent(self, words: List[str], k: int) -> List[str]:
        memo = Counter(words)
        h = []
        result = []

        for key, value in memo.items():
          heappush(h, (-value, key))

        for _ in range(k):
          result.append(heappop(h)[1])

        return result
```

518. Coin change 2
We will iterate each coin first to prevent duplication combination. With this iteration, we will iterate from coin to amount to accumulate nums[i - coin]. In the end, nums[amount] is the amount we want. TC is O(n * amount)

```
class Solution:
    def change(self, amount: int, coins: List[int]) -> int:
        nums = [0] * (amount + 1)
        nums[0] = 1



        for coin in coins:
          for i in range(coin, amount + 1):
              nums[i] += nums[i - coin]
        return nums[amount]
```

54. Spiral Matrix
We will iterate from left to right, top to bottom, right to left, bottom to top. Each time we finish traversal, we will add or deduct that edge. In the middle, we need to check whether left <= right and bottom >= top to prevent two edge has met and the other two not. TC is O(mn)

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        result = []
        if not matrix or not matrix[0]:
```

```
        return result

      top, right, left, bottom = 0, len(matrix[0]) - 1, 0, len(matrix) - 1

      while top <= bottom and left <= right:
        for j in range(left, right + 1):
          result.append(matrix[top][j])

        top += 1

        for i in range(top, bottom + 1):
          result.append(matrix[i][right])

        right -= 1

        if not (top <= bottom and left <= right):
          break

        for j in range(right, left - 1, -1):
          result.append(matrix[bottom][j])

        bottom -= 1

        for i in range(bottom, top - 1, -1):
          result.append(matrix[i][left])
        left += 1
      return result
```

435. Non overlapping intervals

We will sort our intervals first and compare the adjacent intervals, if there is an overlap, we will abandon the interval whose right bound is larger. And use cur one to compare the next one until the end. TC is O(nlogn)

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        if not intervals:
          return 0

        intervals.sort()
        count = 0
        cur = intervals[0]

        for interval in intervals[1:]:
          if cur[1] <= interval[0]:
```

```
            cur = interval
          else:
            count += 1
            if cur[1] > interval[1]:
              cur = interval

        return count
```

599. Minimum index sum of two lists
First, we will use hashmap to remember each restaurant's index in list1, then in list2, we will first check whether this restaurant exists in the hashmap, if it does, we will compare it to the current min_index_sum, if it's less than this one, we will clear our result array to store this one, if equal will append this one to our current result. TC is O(m + n)

```
from collections import defaultdict
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        memo = defaultdict(int)
        result, min_length = [], 2001
        for idx, r in enumerate(list1):
          memo[r] = idx
        for idx, r in enumerate(list2):
          if r in memo:
            if min_length > idx + memo[r]:
              min_length = idx + memo[r]
              result = [r]
            elif min_length == idx + memo[r]:
              result.append(r)
        return result
```