811. Subdomain Visit Count

This is question takes a lot of work. We need to parse time and every subdomain, then use map to store these subdomain and its associated times. In the end, we will iterate through all keys in object and combine the strings together. The TC is O(n*n)

```
const subdomainVisits = (cpdomains) => {
    const calc = {};
    const result = []
    cpdomains.forEach((c) => {
        const words = c.split(' ');
        const subs = words[1].split('.').reverse();
        const num = parseInt(words[0]);
        let subDomain = null;
        subs.forEach((word) => {
            if (!subDomain) {
                subDomain = word;
            } else {
                subDomain = word + '.' + subDomain;
            }
            if (!calc[subDomain]) {
                calc[subDomain] = num;
            } else {
                calc[subDomain] += num;
            }
        });
    });
    for (const key in calc) {
        result.push(calc[key] + ' ' + key);
    }
    return result;
};
```

215. Kth Largest Element in an Array

We could use a heap to maintain k elements, then the minimum one is what we want. TC is nlogn. Here I use a less efficient one, heapify the whole nums, and pop len(nums) - k.

```
const subdomainVisits = (cpdomains) => {
    const calc = {};
    const result = []
    cpdomains.forEach((c) => {
        const words = c.split(' ');
        const subs = words[1].split('.').reverse();
        const num = parseInt(words[0]);
        let subDomain = null;
        subs.forEach((word) => {
            if (!subDomain) {
```

```
            subDomain = word;
        } else {
            subDomain = word + '.' + subDomain;
        }
        if (!calc[subDomain]) {
            calc[subDomain] = num;
        } else {
            calc[subDomain] += num;
        }
    });
});
for (const key in calc) {
    result.push(calc[key] + ' ' + key);
}
return result;
};
```

17. Letter Combinations of a Phone Number

We will iterate every digit and iterate through all characters it represents. Then combine previous one with each of these characters. TC is O(n):

```
var letterCombinations = function(digits) {
    const Table = {'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl', '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'};
    let cur = [''];
    let next = [];
    if (digits === '') {
        return [];
    }
    for (let i = 0; i < digits.length; i++) {
        const words = Table[digits[i]];
        for (let j = 0; j < words.length; j++) {
            for (let k = 0; k < cur.length; k++) {
                next.push(cur[k] + words[j])
            }
        }
        cur = next;
        next = [];
    }
    return cur;
};
```

322. Coin Change

This question is very simple. We only need to use dp to calculate minimum for the current coin number. Memo[cur] = min(Memo[cur - coin]), TC is O(mn).

```
var coinChange = function(coins, amount) {
    const memo = [0];
    for (let i = 0; i < amount; i++) {
        let nextAmount = amount + 1;
        for (let j = 0; j < coins.length; j++) {
            const coin = i + 1 - coins[j];
            if (coin >= 0 && memo[coin] >= 0) {
                if (nextAmount > memo[coin] + 1) {
                    nextAmount = memo[coin] + 1;
                }
            }
        }
        if (nextAmount !== amount + 1) {
            memo.push(nextAmount);
        } else {
            memo.push(-1);
        }
    }
    return memo[amount];
};
```

54. Spiral Matrix

This is quite simple, we just set the boundaries for top, bottom, left, right. Each time we iterate from left to right, top to bottom, right to left, bottom to top. Each time we need to increase or decrease each boundary. Until any time left > right or top > bottom. TC is O(mn)

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix or not matrix[0]:
            return []
        top, bottom, left, right = 0, len(matrix) - 1, 0, len(matrix[0]) - 1
        i, j = 0, 0
        result  = []


        while left <= right and top <= bottom:
            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 (left <= right and top <= bottom):
        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
```