224. Basic Calculator

When it’s ‘ ’, we continue, when it’s a num, we just sum it with other digits. When it’s ‘(’, we will add num to result then push current result and operator to stack and set result and num to zero, mark to 1. When it’s ‘+-’, we will operate previous num and set num to zero and reassign mark new value. When it’s ‘)’, we will pop out operator and previous result, add to the current result. In the end, if num is larger than 0, we will add or deduct it to or from the current result. TC is O(n).

```
class Solution:
    def calculate(self, s: str) -> int:
        stack = []
        mark = 1
        num = 0
        result = 0

        for i in s:
            if i == ' ':
                continue
            if i in '+-':
                result = result + mark * num
                num = 0
                mark = 1 if i == '+' else -1
            elif i == '(':
                if num > 0:
                    result += num
                stack.append(result)
                result = 0
                num = 0
                stack.append(mark)
                mark = 1
            elif i == ')':
                result = result + mark * num

                mark = stack.pop()
                pre = stack.pop()

                result = result * mark + pre
                num = 0
            else:
                num = num * 10 + int(i)
        if num > 0:
            result += mark * num
        return result
```

394. Decode String

We use two stacks to store number and characters, When it’s number, we add it to the stored one. When it’s ‘[’, we will push all previous words and numbers to the stack and set words and numbers. When it’s ‘]’, we pop out the previous words and number, then append new multiplied words to previous words. For normal word, we append it to words directly. The TC is O(n)

```
class Solution:
    def decodeString(self, s: str) -> str:
        Num = '1234567890'
        result = ''
        int_stack = []
        str_stack = []
        num = 0

        for i in s:
            if i in Num:
                num = num * 10 + int(i)
            elif i == '[':
                str_stack.append(result)
                int_stack.append(num)
                result = ''
                num = 0
            elif i == ']':
                temp = result
                result = str_stack.pop()
                num = int_stack.pop()
                result = result + temp * num
                num = 0
            else:
                result += i

        return result
```

127. Word Ladder.

This question is a little tricky. We only need to search words in wordList until we cannot find associated word. We only find next word that exists in wordList so that the whole process could be shorter. The TC is O(n)

```
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        visited = set()
        letters = 'abcdefghijklmnopqrstuvwxyz'
        words = set(wordList)

        def dfs(cur_words, visited, time):
            next_ite = []
            for w in cur_words:
                for i in range(len(w)):
                    for l in letters:
                        new_word = w[:i] + l + w[i + 1:]
                        if new_word in words and new_word not in visited:
                            if new_word == endWord:
                                return time + 1
                            next_ite.append(new_word)
                            visited.add(new_word)
            if next_ite:
                return dfs(next_ite, visited, time + 1)
            else:
                return 0

        return dfs([beginWord], visited, 1)
```

68. Text Justification

We just follow the instruction to operate our strings. Nothing more to say. TC is O(n).

```
class Solution:
    def fullJustify(self, words: List[str], maxWidth: int) -> List[str]:
        result = []
        strings = []
        total_length = 0
        words_length = 0

        def consistWord(my_words, word_length):
            length = len(my_words)
            if length == 1:
                return my_words[0] + ' ' * (maxWidth - word_length)

            mod, rest = divmod(maxWidth - word_length, length - 1)
            my_words = my_words[::-1]
            temp = ''

            for _ in range(rest):
                temp += my_words.pop() + ' ' * (mod + 1)
            for _ in range(length - 1 - rest):
                temp += my_words.pop() + ' ' * mod
            temp += my_words.pop()
            return temp

        for idx, word in enumerate(words):
            length = len(word)
            if total_length == 0:
                total_length = length
            elif total_length + length + 1 <= maxWidth:
                total_length += length + 1
            else:
                result.append(consistWord(strings, words_length))
                total_length = length
                words_length = 0
                strings = []
            words_length += length
            strings.append(word)
            if idx == len(words) - 1:
                temp = ' '.join(strings)
                result.append(temp + ' ' * (maxWidth - len(temp)))
                return result
```

986. Interval List Intersections

This question is very simple. We only need to check weather two interval has intersections, it they have, we just append intersections to result, and then move interval to next one according to which interval's end is smaller. TC is O(m + n).

```
class Solution:
    def intervalIntersection(self, A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
        result = []
        length_A, length_B = len(A), len(B)
        ind_A, ind_B = 0, 0

        while ind_A < length_A and ind_B < length_B:
            start_A, end_A = A[ind_A]
            start_B, end_B = B[ind_B]
            if not (end_A < start_B or end_B < start_A):
                result.append([max(start_A, start_B),min(end_A, end_B)])
            if end_A < end_B:
                ind_A += 1
            elif end_A > end_B:
                ind_B += 1
            else:
                ind_A += 1
                ind_B += 1
        return result
```