241. Different Ways to Add Parentheses
We will use divide and conquer to solve this question. TC is O(len(operation)!)

```
class Solution:
    def diffWaysToCompute(self, input: str) -> List[int]:
        if input.isdigit():
            return [int(input)]
        res = []
        for idx, e in enumerate(input):
            if e in '+-*':
                res1 = self.diffWaysToCompute(input[:idx])
                res2 = self.diffWaysToCompute(input[idx + 1:])
                for a1 in res1:
                    for a2 in res2:
                        res.append(self.helper(a1, a2, e))
        return res

    def helper(self, a1, a2, op):
        if op == '+':
            return a1 + a2
        elif op == '-':
            return a1 - a2
        else:
            return a1 * a2
```

282. Expression Add Operators
We will backtracking to iterate all possible combinations in num. TC is O(3 ** (n - 1)))

```
class Solution:
    def addOperators(self, num: str, target: int) -> List[str]:
        self.target = target
        res = []
        for i in range(1, len(num) + 1):
            if i == 1 or (i > 1 and num[0] != '0'):
                self.dfs(num[i:], num[:i], int(num[:i]), int(num[:i]), res)
        return res

    def dfs(self, num, temp, cur, last, res):
        if not num:
            if cur == self.target:
                res.append(temp)
            return
        for i in range(1, len(num) + 1):
            val = int(num[:i])
            if i == 1 or (i > 1 and num[0] != '0'):
```

```
            self.dfs(num[i:], temp + '+' + str(val), cur + val, val, res)
            self.dfs(num[i:], temp + '-' + str(val), cur - val, -val, res)
            self.dfs(num[i:], temp + '*' + str(val), cur - last + last * val, last * val, res)
```

842. Split Array into Fibonacci Sequence
We will find all combination of first and second number and go to the next one until the end of num. TC is O(10 * 10 + n)

```
class Solution:
    def splitIntoFibonacci(self, S: str) -> List[int]:
        res = []
        self.max_val = 2**31 - 1
        for i in range(1, len(S) - 1):
            prev1 = int(S[:i])
            if prev1 > self.max_val:
                break
            if i == 1 or (i > 1 and S[0] != '0'):
                for j in range(i + 1, len(S)):
                    if j == i + 1 or (j > i + 1 and S[i] != '0'):
                        prev2 = int(S[i:j])
                        if prev1 > self.max_val:
                            break
                        temp = self.dfs([prev1, prev2], prev1, prev2, j, S, res)
                        if temp:
                            return temp
        return []

    def dfs(self, cur, prev1, prev2, cur_idx, S, res):
        if cur_idx == len(S):
            return cur
        num = str(prev1 + prev2)
        if prev1 + prev2 > self.max_val:
            return []
        if num == S[cur_idx:cur_idx+len(num)]:
            temp = self.dfs(cur + [prev1 + prev2], prev2, prev1 + prev2, cur_idx+len(num), S, res)
            if temp:
                return temp
        return []
```

720. Longest Word in Dictionary
We will use Trie and DFS to solve this question. First we will insert words into our trie and then we will use dfs to get all posible longest words we could get and use max_word to records. TC is O(n * len(w))

```
from collections import defaultdict
```

```
class Trie:
    def __init__(self):
        self.trie = {}

    def insert(self, word):
        node = self.trie
        for i in word:
            if i not in node:
                node[i] = {}
            node = node[i]
        node['word'] = True


class Solution:
    def longestWord(self, words: List[str]) -> str:
        trie = Trie()
        self.max_length = 0
        self.max_word = ''
        for word in words:
            trie.insert(word)

        node = trie.trie
        for i, val in node.items():
            if i != 'word' and 'word' in val:
                self.dfs(i, val, 1)
        return self.max_word

    def dfs(self, cur, node, cur_length):
        if self.max_length < cur_length:
            self.max_length = cur_length
            self.max_word = cur
        elif self.max_length == cur_length:
            self.max_word = min(self.max_word, cur)

        for i, val in node.items():
            if i != 'word' and 'word' in val:
                self.dfs(cur + i, val, cur_length + 1)
```

648. Replace Words

We will use Trie and search every word’s root in sentence, if its root exists, we will replace that, TC is O(n* len(w))

```
class Trie:
    def __init__(self):
```

```
        self.trie = {}

    def insert(self, word):
        node = self.trie
        for i in word:
            if i not in node:
                node[i] = {}
            node = node[i]
        node['word'] = True

    def findRoot(self, word):
        node = self.trie
        for i, e in enumerate(word):
            if e not in node:
                return word
            if 'word' in node[e]:
                return word[:i + 1]
            node = node[e]
        return word

class Solution:
    def replaceWords(self, dict: List[str], sentence: str) -> str:
        trie = Trie()
        for word in dict:
            trie.insert(word)
        return ' '.join(map(lambda a: trie.findRoot(a), sentence.split()))
```