8. String to Integer (atoi)

```
class Solution:
    def helper(self,s,i,ans):
        if i < len(s) and s[i] in "0123456789":
            return self.helper(s,i+1,ans+s[i])
        return ans

    def myAtoi(self, s: str) -> int:
        s = s.strip()
        if not s:return 0
        ans = 1
        if s[0] == "-":
            ans = -1
            s = s[1:]
        elif s[0] == "+":
            s = s[1:]
        temp = self.helper(s,0,"")
        if not temp:return 0
        ans = int(temp) * ans
        if ans < -2**31 :
            ans = -2**31
        elif ans > (2**31) - 1:
            ans = (2**31) - 1
        return ans
```

17. Letter Combinations of a Phone Number

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:return []
        my_dict = {
            "2": "abc",
            "3": "def",
            "4": "ghi",
            "5": "jkl",
            "6": "mno",
            "7": "pqrs",
            "8": "tuv",
            "9": "wxyz"
        }
        res = []
        def backtrack(ind,temp):
            if ind >= len(digits):
                res.append(temp)
                return
            digit = digits[ind]
            for j in range(len(my_dict[digit])):
                backtrack(ind+1,temp+my_dict[digit][j])

        backtrack(0,"")
        return res
```

22. Generate Parentheses

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        def recursion(open,close,temp):
            if len(temp)== n*2:
                res.append(temp)
                return
            if open < n:
                recursion(open+1,close,temp+"(")
            if close < open:
                recursion(open,close+1,temp+")")
        recursion(1,0,"(")
        return res
```

37. Sudoku Solver

```
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
        def is_valid(board,row,col,ch):
            for i in range(0,9):
                if board[row][i] == ch or board[i][col] == ch:return False
                if board[3*(row//3)+i//3][3*(col//3)+i%3] == ch:return False
            return True

        def solve(board):
            for i in range(9):
                for j in range(9):
                    if board[i][j] == ".":
                        for k in range(1,10):
                            if is_valid(board,i,j,str(k)):
                                board[i][j] = str(k)
                                if solve(board):
                                    return True
                                board[i][j] = "."
                        return False
            return True
        solve(board)
```

39. Combination Sum

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[in
t]]:
        res = []
        def recursion(ind,temp,summ):
            if summ == target:
                res.append(temp[:])
                return
            if ind >= len(candidates) or summ > target:
                return
            temp.append(candidates[ind])
            recursion(ind,temp,summ+candidates[ind])
            temp.pop()
            recursion(ind+1,temp,summ)
        recursion(0,[],0)
        return res
```

40. Combination Sum II

```
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[in
t]]:
        res= []
        candidates.sort()
        def recursion(start,summ,subset):
            if summ == target:
                res.append(subset[:])
                return
            if summ > target or start >= len(candidates):
                return

            for ind in range(start,len(candidates)):
                if ind > start and candidates[ind] == candidates[ind-1]:
                    continue
                subset.append(candidates[ind])
                recursion(ind+1,summ+candidates[ind],subset)
                subset.pop()
        recursion(0,0,[])
        return res
```

50. Pow(x, n)

If n is even: $x^n = x^{n/2} \cdot x^{n/2}$

If n is odd: $x^n = x^{\lfloor n/2 \rfloor} \cdot x^{\lfloor n/2 \rfloor} \cdot x$

```
class Solution:
    def helper(self,x,n):
        if n == 0:return 1
        half = self.helper(x,n//2)
        if n%2 == 0:
            return half * half
        else:
            return half*half*x

    def myPow(self, x: float, n: int) -> float:
        ans  = self.helper(x,n if n > 0 else n*-1)
        return ans if n>0 else 1/ans
```

78. Subsets

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        def recursion(idx,temp):
            if idx >= len(nums):
                res.append(temp.copy())
                return
            temp.append(nums[idx])
            recursion(idx+1,temp)
            temp.pop()
            recursion(idx+1,temp)
        recursion(0,[])
        return res
```

79. Word Search

Get better space coplexity by using a temp = board[i][j] board[i][j] = "#" and then again asign the temp

board = [i][j]

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m = len(board)
        n = len(board[0])
        def recursion(i,j,index):
            if index >= len(word):return True
            if not (0 <= i < m) or not (0<= j < n) or (i,j) in vis or board[i][j] !
= word[index]:return False
            vis.add((i,j))
            if recursion(i+1,j,index+1) or recursion(i,j+1,index+1) or recursion
(i,j-1,index+1) or recursion(i-1,j,index+1):
                return True
            vis.remove((i, j))
            return False
        for i in range(m):
            for j in range(n):
                vis = set()
                if board[i][j] == word[0] :
                    if recursion(i,j,0):
                        return True
        return False
```

---

90. Subsets II

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        def recursion(ind,subset):
            res.append(subset[:])
            if ind == len(nums):
                return
            for i in range(ind,len(nums)):
                if i > ind and nums[i] == nums[i-1]:
                    continue
                subset.append(nums[i])
                recursion(i+1,subset)
                subset.pop()
        recursion(0,[])
        return res
```

131. Palindrome Partitioning

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        #https://leetcode.com/problems/palindrome-partitioning/
        res = []
        def backtrack(start,temp):
            if start == len(s):
                res.append(temp[:])
                return
            for end in range(start+1,len(s)+1):
                if s[start:end] == s[start:end][::-1]:
                    backtrack(end,temp+ [s[start:end]])

        backtrack(0,[])
        return res
```

216. Combination Sum III

```
use style 2 as that is preferred [for loop combination good for maintaining order a
nd duplicates ]

class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        res = []
        def recursion(ind,summ,subset):
            if n == summ and len(subset) == k:
                res.append(subset[:])
                return
            if len(subset) >= k or summ > n or ind > 9:
                return

            subset.append(ind)
            recursion(ind+1,summ+ind,subset)
            subset.pop()
            recursion(ind+1,summ,subset)
        recursion(1,0,[])
        return res


from typing import List

class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        result = []

        def backtrack(start: int, current_sum: int, path: List[int]):
            # Base case: valid combination
            if len(path) == k and current_sum == n:
                result.append(path[:])
                return
            # Pruning: stop if invalid path
            if len(path) > k or current_sum > n:
                return

            for i in range(start, 10):  # Only digits 1 through 9
                path.append(i)
                backtrack(i + 1, current_sum + i, path)
                path.pop()

        backtrack(1, 0, [])
        return result
```

282. Expression Add Operators

```
from typing import List

class Solution:
    def addOperators(self, num: str, target: int) -> List[str]:
        res = []

        def backtrack(index: int, path: str, eval_val: int, prev_num: int):
            if index == len(num):
                if eval_val == target:
                    res.append(path)
                return

            for i in range(index, len(num)):
                # Skip numbers with leading zeros
                if i != index and num[index] == '0':
                    break

                curr_str = num[index:i + 1]
                curr_num = int(curr_str)

                if index == 0:
                    # First number, start expression
                    backtrack(i + 1, curr_str, curr_num, curr_num)
                else:
                    # '+'
                    backtrack(i + 1, path + "+" + curr_str, eval_val + curr_num, cu
rr_num)
                    # '-'
                    backtrack(i + 1, path + "-" + curr_str, eval_val - curr_num, -c
urr_num)
                    # '*'
                    backtrack(i + 1, path + "*" + curr_str,
                              eval_val - prev_num + (prev_num * curr_num),
                              prev_num * curr_num)

        backtrack(0, "", 0, 0)
        return res
```

1922. Count Good Numbers

https://leetcode.com/problems/count-good-numbers/solutions/6645467/beats-super-easy-beginners-java-c-c-python-javascript-dart/ (https://leetcode.com/problems/count-good-numbers/solutions/6645467/beats-super-easy-beginners-java-c-c-python-javascript-dart/)

```
class Solution:
    def countGoodNumbers(self, n: int) -> int:
        mod = 10**9 + 7
        even = (n + 1) // 2
        odd = n // 2
        return (pow(5, even, mod) * pow(4, odd, mod)) % mod
```
