47. Permutations II
We will use iteration to insert each num to each previous arrays. When we encounter same element, we will break to prevent duplication. TC is O(n ^ 2)

```
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        ans = [[]]
        for num in nums:
            cur = []
            for l in ans:
                for i in range(len(l) + 1):
                    cur.append(l[:i] + [num] + l[i:])
                    if i < len(l) and l[i] == num:
                        break
            ans = cur
        return ans
```

47. Permutations II

```
from collections import Counter
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        res = []
        counter = Counter(nums)
        def backTrack(cur):
            if len(cur) == len(nums):
                res.append(cur[:])
                print(res)
                return
            for k, v in counter.items():
                if v <= 0:
                    continue
                cur.append(k)
                counter[k] -= 1
                backTrack(cur)
                cur.pop()
                counter[k] += 1
        backTrack([])
        return res
```

784. Letter Case Permutation
We will use dfs to combine all possible lower or upper letters. TC is O(n)

```
class Solution:
    def letterCasePermutation(self, S: str) -> List[str]:
        res = []
```

```
        def dfs(cur, idx):
            if len(cur) == len(S):
                res.append(cur)
                return
            if S[idx] in '1234567890':
                dfs(cur + S[idx], idx + 1)
            else:
                dfs(cur + S[idx].lower(), idx + 1)
                dfs(cur + S[idx].upper(), idx + 1)
        dfs('', 0)
        return res
```

301. Remove Invalid Parentheses

We will get the number of left and right parentheses that we need to delete in s. Then we will use dfs to delete ‘(’ or ‘)’ or not and move to next stage until the end of s. We will append the valid string to res. TC is O(2 ** l)

```
class Solution:
    def removeInvalidParentheses(self, s: str) -> List[str]:
        l, r = 0, 0
        r1, l1 = self.getRemovePa(s, '(', ')')
        l2, r2 = self.getRemovePa(s[::-1], ')', '(')
        l = max(l1, l2)
        r = max(r1, r2)
        self.res = []
        self.dfs(l, r, s, 0, 0)
        return self.res

    def getRemovePa(self, s, l, r):
        count = 0
        res = 0
        for i in s:
            if i == l:
                count += 1
            elif i == r:
                if count == 0:
                    res += 1
                    continue
                count -= 1
        return res, count

    def dfs(self, l, r, s, count, idx):
        for i in range(idx, len(s)):
            if s[i] == '(':
```

```
            if i > idx and s[i - 1] == '(':
                count += 1
                continue
            if l > 0:
                self.dfs(l - 1, r, s[:i] + s[i + 1:], count, i)
            count += 1
        elif s[i] == ')':
            if i > idx and s[i - 1] == ')':
                if count > 0:
                    count -= 1
                    continue
                else:
                    return
            if r > 0 and count >= 0:
                self.dfs(l, r - 1, s[:i] + s[i + 1:], count, i)
            count -= 1
            if count < 0:
                return
    if l == 0 and r == 0 and count == 0:
        self.res.append(s)
```

51. N-Queens

We will use dfs to iterate through all possible combinations and append it to res once it achieves to the end of n. TC is O(n ** n)

```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        cols = set()
        dia1 = set()
        dia2 = set()
        res = []
        def dfs(i, cur):
            if i > n:
                return
            if i == n:
                matrix = []
                for i in cur:
                    a = ['.'] * n
                    a[i] = 'Q'
                    matrix.append(''.join(a))
                res.append(matrix)
            for j in range(0, n):
                if j not in cols and i + j not in dia1 and i - j not in dia2:
                    cols.add(j)
```

```
            dia1.add(i + j)
            dia2.add(i - j)
            cur.append(j)
            dfs(i + 1, cur)
            cols.remove(j)
            dia1.remove(i + j)
            dia2.remove(i - j)
            cur.pop()
    dfs(0, [])
    return res
```