77. Combinations
We will use dfs and iterate all elements from 1 to n, once current array's length is equal to k and append current array to result. TC is O(n ** k)

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []
        def dfs(cur, idx):
            if len(cur) == k:
                res.append(cur)
            else:
                for i in range(idx, n + 1):
                    dfs(cur + [i], i + 1)
        dfs([], 1)
        return res
```

77. Combinations
We will use backtracking to solve this question. We will append x to stack until len(stack) == k or x is too large for later number to add. Because we should make our arrays in increasing order to prevent duplication. At that point, we need to backtrack, pop our stack and accumulate our x. Repeat this process until stack is empty. TC is O(n**2)

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        stack = []
        res = []
        x = 1
        while True:
            l = len(stack)
            if l == k:
                res.append(stack[:])
            if l == k or x > n - k + l + 1:
                if not stack:
                    return res
                x = stack.pop() + 1
            else:
                stack.append(x)
                x += 1
```

78. Subsets
We will iterate through all elements in nums and append all its previous arrays. TC is O(n * *2)

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = [[]]
        for num in nums:
```

```
            cur = []
            for e in res:
                cur.append(e + [num])
            res = res + cur
        return res
```

90. Subsets II

We will iterate through our sorted nums, once there is a seen num, we will only append it to the previous one rather than res. TC is O(n ** 2)

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        res = [[]]
        seen = set()
        cur = []
        nums.sort()
        for num in nums:
            if num in seen:
                cur = [e + [num] for e in cur]
            else:
                seen.add(num)
                cur = [e + [num] for e in res]
            res = res + cur
        return res
```

216. Combination Sum III

We will use dfs to get our combination, every time when we pick one number, we will pick the rest from the right side to prevent duplication. TC is O(n * k)

```
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        res = []
        if not k or not n:
            return res
        def dfs(cur, idx, cur_sum):
            if len(cur) > k or cur_sum > n:
                return
            if cur_sum == n and len(cur) == k:
                res.append(cur)
                return
            for i in range(idx, 10):
                dfs(cur + [i], i + 1, cur_sum + i)
        dfs([], 1, 0)
        return res
```