## 77. Combinations

8125 218 Add to List Share

Given two integers `n` and `k`, return *all possible combinations of* `k` *numbers chosen from the range* `[1, n]`.

You may return the answer in **any order**.

```java
class Solution {
    public void f(int i, List<List<Integer>> res, List<Integer> ds, int k, int n) {
        if (ds.size() == k) {
            res.add(new ArrayList<>(ds));
            return;
        }

        if (i == n) {
            return;
        }

        ds.add(i + 1);
        f(i + 1, res, ds, k, n);
        ds.remove(ds.size() - 1);
        f(i + 1, res, ds, k, n);
    }

    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> res = new ArrayList<>();
        f(0, res, new ArrayList<>(), k, n);
        return res;
    }
}
```

## 40. Combination Sum II



Given a collection of candidate numbers ( `candidates` ) and a target number ( `target` ), find all unique combinations in `candidates` where the candidate numbers sum to `target` .

Each number in `candidates` may only be used **once** in the combination.

**Note:** The solution set must not contain duplicate combinations.

```
class Solution {
    public void generate(int idx, int target, int[] nums, ArrayList<Integer> vec, List<List<Integer>> res) {
        if (target == 0) {
            res.add(new ArrayList<>(vec));
            return;
        }
        for (int i = idx; i < nums.length; i++) {
            if (i > idx && nums[i] == nums[i - 1]) {
                continue;
            }
            if (nums[i] > target) {
                break;
            }
            vec.add(nums[i]);
            generate(i + 1, target - nums[i], nums, vec, res);
            vec.remove(vec.size() - 1);
        }
    }

    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(candidates);
        generate(0, target, candidates, new ArrayList<>(), res);
        return res;
    }
}
```

## 216. Combination Sum III

5860 109 Add to List Share

Find all valid combinations of `k` numbers that sum up to `n` such that the following conditions are true:

- Only numbers `1` through `9` are used.
- Each number is used **at most once**.

Return *a list of all possible valid combinations*. The list must not contain the same combination twice, and the combinations may be returned in any order.

```
class Solution {
    public void f(int i, List<List<Integer>> res, List<Integer> ds, int k, int target) {
        if (i == 9) {
            if (target == 0 && ds.size() == k) {
                res.add(new ArrayList<>(ds));
            }
            return;
        }

        if ((i + 1) <= target) {
            ds.add(i + 1);
            f(i + 1, res, ds, k, target - (i + 1));
            ds.remove(ds.size() - 1);
        }
        f(i + 1, res, ds, k, target);
    }

    public List<List<Integer>> combinationSum3(int k, int n) {
        List<List<Integer>> res = new ArrayList<>();
        f(0, res, new ArrayList<>(), k, n);
        return res;
    }
}
```

## 78. Subsets

16677 268 Add to List Share

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

```
class Solution {
    public void subset(int index, int[] nums, List<Integer> ds, List<List<Integer>> res) {
        if (index == nums.length) {
            res.add(new ArrayList<>(ds));
            return;
        }
        ds.add(nums[index]);
        subset(index + 1, nums, ds, res);
        ds.remove(ds.size() - 1);
        subset(index + 1, nums, ds, res);
    }

    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        subset(0, nums, new ArrayList<>(), res);
        return res;
    }
}
```