39. Combination Sum

Given a **set** of candidate numbers (`candidates`) **(without duplicates)** and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

The **same** repeated number may be chosen from `candidates` unlimited number of times.

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> results = new ArrayList<>();
        if (candidates == null || candidates.length == 0)
            return results;

        int[] nums = removeDuplicates(candidates);

        // backtrack
        helper(nums, 0, new ArrayList<Integer>(), target, results);
        return results;
    }


    private void helper(int[] nums, int startIndex, List<Integer> combination,
int target,
                        List<List<Integer>> results) {

        if (target == 0) {
            results.add(new ArrayList<Integer>(combination));
            return;
        }

        for (int i = startIndex; i < nums.length; i++) {
            if (nums[i] > target)
                break;

            combination.add(nums[i]); // [1] -> [1, 2]

            // start from [1, 2], new target is remained target
            helper(nums, i, combination, target - nums[i], results);

            // find all from [1, 2], return to [1,2] -> [1]
            combination.remove(combination.size() - 1);
        }
    }


    private int[] removeDuplicates(int[] candidates) {
        Arrays.sort(candidates);

        int index = 0;
        for (int i = 0; i < candidates.length; i++) {
            if (candidates[i] != candidates[index])
                candidates[++index] = candidates[i];
        }
```

```
        int[] nums = new int[index + 1];
        for (int i = 0; i < index + 1; i++) {
            nums[i] = candidates[i];
        }

        return nums;
    }
}
```