39. Combination Sum

Medium

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

three similar solutions: keep track of
- index of candidates list → i
- current permutation → current
- sum of items in permutation → total

candidates = [2,3,5]
target = 8

```
class Solution:
    def combinationSum1(self, candidates: List[int], target: int) -> List[List[int]]:
        answer = []
        permutation = []

        def backtrack(i):
            if sum(permutation) == target:
                answer.append(permutation[:])
                return
            if i >= len(candidates) or sum(permutation) > target:
                return

            permutation.append(candidates[i])
            backtrack(i)

            permutation.pop()
            backtrack(i + 1)

        backtrack(0)

        return answer
```

i = 0

print(permutation)

|| if (i ≥ len(candidates) or sum(permutation) > target)

candidates[0] → 2

i = 0

permutation
[2]
[2,2]
[2,2,2]
[2,2,2,2] ⇒ sum([2,2,2,2]) == 8 (target)

.pop() [2,2,2,~~2~~]

backtrack(i+1)
[2,2,2,3]
[2,2,2,~~3~~]
[2,2,~~2~~]
[2,2,3]
[2,2,3,3]
[2,2,3,~~3~~]
[2,2,3,5]
⋮

```
[]
[2]
[2, 2]
[2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2]
[2, 2, 2, 3]
[2, 2, 2]
[2, 2, 2, 5]
[2, 2, 2]
[2, 2]
[2, 2, 3]
[2, 2, 3, 3]
[2, 2, 3]
[2, 2, 3, 5]
[2, 2, 3]
[2, 2]
[2, 2, 5]
[2, 2]
[2]
[2, 3]
[2, 3, 3]
[2, 3]
[2, 3, 5]
[2, 3]
[2]
[2, 5]
[2, 5, 5]
[2, 5]
[2]
[]
[3]
[3, 3]
[3, 3, 3]
[3, 3]
[3, 3, 5]
[3, 3]
[3]
[3, 5]
[3]
[]
[5]
[5, 5]
[5]
[]
```

```
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        answer = []

        def backtrack(i, current):
            if sum(current) == target:
                answer.append(current[:])
                return
            if i >= len(candidates) or sum(current) > target:
                return

            current.append(candidates[i])
            backtrack(i, current)

            current.pop()
            backtrack(i + 1, current)

        backtrack(0, [])

        return answer
```

```
class Solution:
    def combinationSum3(self, candidates: List[int], target: int) -> List[List[int]]:
        answer = []

        def backtrack(i, current, total):
            if total == target:
                answer.append(current.copy())
                return
            if i >= len(candidates) or total > target:
                return

            current.append(candidates[i])
            backtrack(i, current, total + candidates[i])

            current.pop()
            backtrack(i + 1, current, total)

        backtrack(0, [], 0)

        return answer
```