39. Combination Sum

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]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

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

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

Constraints:

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 40`

39. Combination Sum

```
/*
   Time complexity: O(2^target)
   Space complexity: O(target+n)
*/

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution {
public:
   vvi combinationSum(vi& candidates, int target){
      int n=candidates.size();
      vvi ans;

      auto solve=[&](int i,vi& tmp, int sum,auto& self)->void{
         if(sum==0) {
            ans.push_back(tmp);
            return;
         }

         if(i>=n || sum<0) return;

         // Include candidates[i]
         tmp.push_back(candidates[i]);

         // Explore
         self(i,tmp,sum-candidates[i],self);

         // Exclude candidates[i]
         tmp.pop_back();

         self(i+1,tmp,sum,self);
      };

      vi tmp;

      solve(0,tmp,target,solve);

      return ans;
   }
};
```

39. Combination Sum

```
/*
   Time complexity: O(2^target)
   Space complexity: O(target+n)
*/

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution {
   public:
      vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
         int n=candidates.size();
         vvi ans;

         auto solve=[&](int i,vi& tmp, int sum,auto& self)->void{
            if(sum==0) {
               ans.push_back(tmp);
               return;
            }

            if(sum<0) return;

            for(int j=i;j<n;++j){
               if(candidates[j]>target) continue;

               // Include candidates[j]
               tmp.push_back(candidates[j]);

               // Explore
               self(j,tmp,sum-candidates[j],self);

               // Exclude candidates[j]
               tmp.pop_back();
            }
         };

         vi tmp;

         solve(0,tmp,target,solve);

         return ans;
      }
};
```