9. LC 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.

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

arr [] = [1, 1, 1, 2, 2]   target = 4

o/p [1, 1, 2
2, 2]

(1) Brute force:

f (ind, target, ds)

✓ ds.add(a[ind])   ✗

f (ind+1, target - a[ind], ds) // pick   f (ind+1, target, ds) // Not pick

if (a[ind] <= target)

Use hashset so that duplicates combination are not stored.

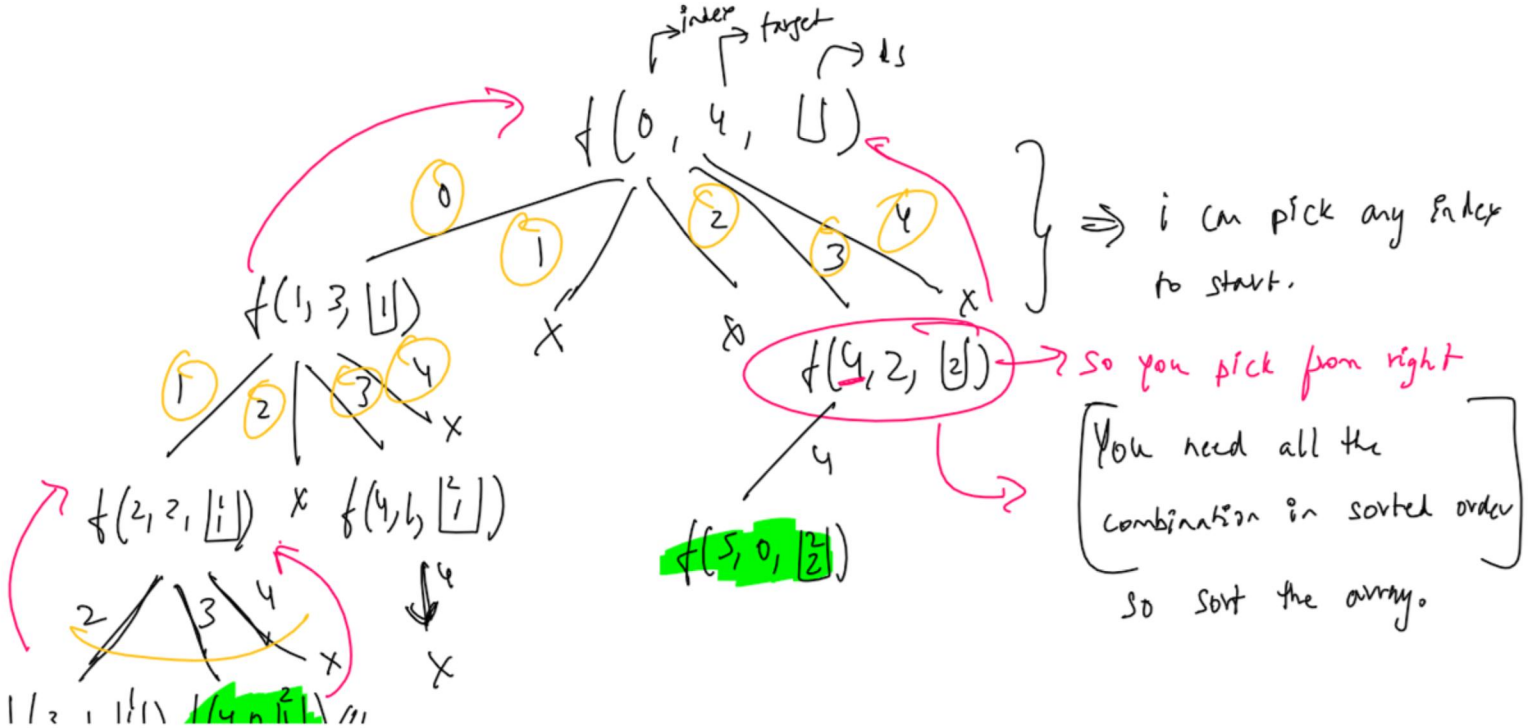
```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, Set<List<Integer>> ans, List<Integer> ds) {
        if(ind == arr.length) {
            if(target == 0) {
                ans.add(new ArrayList<>(ds));
            }
            return;
        }

        if(arr[ind] <= target) {
            ds.add(arr[ind]);
            findCombinations(ind+1, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        findCombinations(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        Set<List<Integer>> ans = new ArrayList<>();
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        // hashSet can be converted to a list of list, then return the list
        return ans;
    }
}
```

$T.C \Rightarrow O(2^n * k) * \log^{setsize}$

(2) Optimised Approach :

1) Instead of using pick and not pick, we will try to pick subsequences.

$arr[] = [1, 1, 1, 2, 2]$ (indices 0 1 2 3 4) target = 4



X

f(3, 1, [1 1]) f(4, 0, [2 1]) // base case

3 4

X X

↳ Reached the target '0'.

→ target value less than array values so we can't proceed. (arr[i] > target)

O/P ⇒ 1 1 2

2 2

⇒ Ans

* Carry ans because the result of ds will store in it

f(ind, target, ds, ans)

loop (ind → n-1) → [i] // checking 0 to n-1

1 to n-1

marked in yellow color.

if (arr[i] > target)
break;

✓ ds.add (arr[i])

f(i+1, target - arr[i], ds, arr)

ds. remove (arr [i])

X No recursion call

base case

if (target == 0)
ans. add (ds)

T.C ⇒ $O(2^n \times k)$ S.C ⇒ kx

Java

Autocomplete

```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds){
        // base condition
        if(target == 0){
            ans.add(new ArrayList<>(ds));
            return;
        }

        for(int i = ind; i < arr.length; i++){
            if(i > ind && arr[i] == arr[i - 1]) continue; // arr[i] == arr[i - 1] to check duplicates
            // i > ind to check because if the first element to decide pick or not for combination next, you put that check
            if(arr[i] > target) break;

            ds.add(arr[i]);
            findCombinations(i + 1, arr, target - arr[i], ans, ds);
            ds.remove(ds.size() - 1);
        }
    }

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

C++

Autocomplete

```cpp
class Solution {
public:
    void findCombination(int ind, int target, vector < int > & arr, vector < vector < int >> & ans, vector < int > & ds)    {
      if (target == 0) {
        ans.push_back(ds);
        return;
      }
      for (int i = ind; i < arr.size(); i++) {
        if (i > ind && arr[i] == arr[i - 1]) continue;
        if (arr[i] > target) break;
        ds.push_back(arr[i]);
        findCombination(i + 1, target - arr[i], arr, ans, ds);
        ds.pop_back();
      }
    }
    vector < vector < int >> combinationSum2(vector < int > & candidates, int target) {
      sort(candidates.begin(), candidates.end());
      vector < vector < int >> ans;
      vector < int > ds;
      findCombination(0, target, candidates, ans, ds);
      return ans;
    }
};
```