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.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

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

**A solution set is:**

[

[7],

[2,2,3]

]

**Example 2:**

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

**A solution set is:**

[

  [2,2,2,2],

  [2,3,3],

  [3,5]

]

1. **class** Solution {
2. **public** List<List<Integer>> combinationSum(**int**[] candidates, **int** target) {
3. List<List<Integer>> res = **new** ArrayList<>();
4. Arrays.sort(candidates);
5. helper(candidates, target, res, **new** ArrayList(),0);
6. **return** res;
7. }
9. **public** **void** helper(**int**[] candidates, **int** target, List<List<Integer>> res, List<Integer> subList,**int** index) {
10. **if** (target == 0) {
11. res.add(**new** ArrayList(subList));
12. } **else** {
13. **for** (**int** i = index; i < candidates.length; i++) {
14. **if** (candidates[i]  > target) **continue**;
15. subList.add(candidates[i]);
16. helper(candidates, target - candidates[i], res, subList, i);
17. subList.remove(subList.size() - 1);
18. }
19. }
20. }
21. }

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 sums to target.

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

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

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

**A solution set is:**

[

[1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6]

]

**Example 2:**

**Input:** candidates = [2,5,2,1,2], target = 5,

**A solution set is:**

[

  [1,2,2],

  [5]

]

1. **class** Solution {
2. **public** List<List<Integer>> combinationSum2(**int**[] candidates, **int** target) {
3. List<List<Integer>> res = **new** ArrayList<>();
4. Arrays.sort(candidates);
5. helper(candidates, target, res, **new** ArrayList(), 0);
6. **return** res;
7. }
9. **public** **void** helper(**int**[] candidates, **int** target, List<List<Integer>> res, List<Integer> subList, **int** index) {
10. **if** (target == 0) {
11. res.add(**new** ArrayList(subList));
12. } **else** {
13. **for** (**int** i = index; i < candidates.length; i++) {
14. **if** (candidates[i] > target) **continue**;
15. **if** (i != index && candidates[i] == candidates[i - 1]) **continue**;
16. subList.add(candidates[i]);
17. helper(candidates, target - candidates[i], res, subList, i + 1);
18. subList.remove(subList.size() - 1);
20. }
21. }
22. }
23. }

216. Combination Sum III

Find all possible combinations of ***k*** numbers that add up to a number ***n***, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

**Note:**

* All numbers will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

**Input:** ***k*** = 3, ***n*** = 7

**Output:** [[1,2,4]]

**Example 2:**

**Input:** ***k*** = 3, ***n*** = 9

**Output:** [[1,2,6], [1,3,5], [2,3,4]]

1. **class** Solution {
2. **public** List<List<Integer>> combinationSum3(**int** k, **int** n) {
3. List<List<Integer>> res = **new** ArrayList<>();
4. helper(k, n, res, **new** ArrayList(), 1);
5. **return** res;
6. }
8. **public** **void** helper(**int** k, **int** n, List<List<Integer>> res, List<Integer> subList, **int** index) {
9. **if** (n < 0) **return**;
10. **if** (k == 0 && n == 0) {
11. res.add(**new** ArrayList(subList));
12. } **else** {
13. **for** (**int** i = index; i <= Math.min(n,9); i++) {
14. subList.add(i);
15. helper(k - 1, n - i, res, subList, i + 1);
16. subList.remove(subList.size() - 1);
17. }
18. }
19. }
20. }

46. Permutations

Given a collection of **distinct** integers, return all possible permutations.

**Example:**

**Input:** [1,2,3]

**Output:**

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

1. **class** Solution {
2. **public** List<List<Integer>> permute(**int**[] nums) {
3. List<List<Integer>> res = **new** ArrayList<>();
4. helper(nums, res, **new** ArrayList(), **new** HashSet(),0);
5. **return** res;
6. }
8. **private** **void** helper(**int**[] nums, List<List<Integer>> res, List<Integer> subList, Set<Integer> set, **int** index) {
9. **if** (index == nums.length) {
10. res.add(**new** ArrayList(subList));
11. **return**;
12. }
13. **for** (**int** i = 0; i < nums.length; i++) {
14. **if** (set.contains(nums[i])) **continue**;
15. subList.add(nums[i]);
16. set.add(nums[i]);
17. helper(nums, res, subList, set, index+1);
18. set.remove(nums[i]);
19. subList.remove(subList.size()-1);
20. }
21. }
22. }

47. Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

**Example:**

**Input:** [1,1,2]

**Output:**

[

[1,1,2],

[1,2,1],

[2,1,1]

]

1. **class** Solution {
2. **public** List<List<Integer>> permuteUnique(**int**[] nums) {
3. List<List<Integer>> res = **new** ArrayList<>();
4. Arrays.sort(nums);
5. helper(nums, res, **new** ArrayList(), **new** **boolean**[nums.length]);
6. **return** res;
7. }
9. **public** **void** helper(**int**[] nums, List<List<Integer>> res, List<Integer> sub, **boolean**[] used) {
10. **if** (sub.size() == nums.length) {
11. res.add(**new** ArrayList(sub));
12. } **else** {
13. **for** (**int** i = 0; i < nums.length; i++) {
15. **if** ((i > 0 && nums[i] == nums[i-1] && !used[i-1] ) || used[i]) **continue**;
16. sub.add(nums[i]);
17. used[i] = **true**;
18. helper(nums, res, sub, used);
19. sub.remove(sub.size() - 1);
20. used[i] = **false**;
21. }
22. }
23. }
24. }