

■ Subsets II — Handling Duplicates

Same include/exclude pattern — but skip duplicate branches

● Recursion

■ MEDIUM

■ What You Will Learn

- How sorting enables duplicate detection in recursion
- The 'skip duplicate at same level' technique
- Why we skip at the same recursion level, not across levels
- Backtracking with a 'choose, explore, unchoose' template

■ Intuition First

Input [1,2,2]: without handling duplicates, you'd generate {2} twice (from the first 2 and the second 2). The fix: SORT the array first. Then, at each recursion level, if the current element equals the previous one, SKIP it — you've already explored that branch.

■ Key Rule

After sorting: if `arr[i] == arr[i-1]` AND we're at the same recursion level (not after including the previous), skip `arr[i]`. This eliminates duplicate subsets without using a HashSet.

■ Visual — [1,2,2] sorted

Tree for [1,2,2] – include/exclude with backtracking:

```
start → []
  include 1 → [1]
    include 2 → [1,2]
      include 2 → [1,2,2] ✓
      exclude 2 → [1,2] ✓
    exclude 2 → [1] (skip next '2', it's same as prev)
      SKIP (arr[i]==arr[i-1] at same level)
  exclude 1 → []
    include 2 → [2]
      include 2 → [2,2] ✓
      exclude 2 → [2] ✓
    exclude 2 → [] (skip next 2)
      SKIP
```

Result: [], [1], [1,2], [1,2,2], [2], [2,2]

■ Java Code

```
1 class Solution {
```

```

2   public List<List<Integer>> subsetsWithDup(int[] nums) {
3       Arrays.sort(nums); // MUST sort to detect duplicates
4       List<List<Integer>> result = new ArrayList<>();
5       backtrack(nums, 0, new ArrayList<>(), result);
6       return result;
7   }
8
9   private void backtrack(int[] nums, int start,
10      List<Integer> curr, List<List<Integer>> result) {
11      result.add(new ArrayList<>(curr)); // Add current subset
12
13      for (int i = start; i < nums.length; i++) {
14          // Skip duplicate at same recursion level
15          if (i > start && nums[i] == nums[i-1]) continue;
16
17          curr.add(nums[i]); // Choose
18          backtrack(nums, i+1, curr, result); // Explore
19          curr.remove(curr.size()-1); // Unchoose
20      }
21  }
22  }

```

■ Time & Space Complexity

Type	Value	What It Means
Time	$O(2^n)$	Worst case all unique: 2^n subsets
Space	$O(n)$	Recursion depth = n , temp list = n

■ Common Mistakes

■ Wrong Skip Condition

Using `if(i>0 && nums[i]==nums[i-1])` instead of `if(i>start && ...)`. The first skips across ALL levels. The second only skips at the SAME level. This distinction is critical!

■ Forgetting to Sort

Without sorting, equal elements might not be adjacent and you can't detect duplicates this way.

■ Pro Tips

■ The Backtracking Template

Choose → Explore → Unchoose. This 3-step template works for Subsets, Permutations, Combination Sum, N-Queens. Memorize it. It's the Swiss Army knife of recursion.

■ Practice Problems

#	Problem	Diff
1	Subsets II (LeetCode 90)	Medium
2	Subsets I (LeetCode 78)	Medium
3	Combination Sum II (LeetCode 40)	Medium
4	Palindrome Partitioning (LeetCode 131)	Medium

■ Quick Revision

✓	Sort input array FIRST to group duplicates
✓	Skip: if($i > \text{start}$ && $\text{nums}[i] == \text{nums}[i-1]$) continue
✓	Template: add to curr → recurse → remove from curr
✓	Add snapshot (new ArrayList<>(curr)) at each step
✓	Time $O(2^n)$, Space $O(n)$