# **Leetcode 90 – Subsets**

## Problem Understanding

**Leetcode 90. Subsets II**  
Given an integer array nums that may contain **duplicates**, return all possible **unique subsets** (the power set).

* The solution must **not contain duplicate subsets**.
* The input array may have **repeated numbers** (unlike Leetcode 78).
* You must **handle duplicates carefully** using **sorting + skipping** logic.

## Optimized Java Solution (Backtracking with Skip-Duplicates)

import java.util.\*;

public class Solution {

public List<List<Integer>> subsetsWithDup(int[] nums) {

List<List<Integer>> result = new ArrayList<>();

Arrays.sort(nums); // sort to bring duplicates together

backtrack(0, nums, new ArrayList<>(), result);

return result;

}

private void backtrack(int start, int[] nums, List<Integer> path, List<List<Integer>> result) {

result.add(new ArrayList<>(path)); // add current subset

for (int i = start; i < nums.length; i++) {

if (i > start && nums[i] == nums[i - 1]) continue; // skip duplicate at same level

path.add(nums[i]);

backtrack(i + 1, nums, path, result); // explore

path.remove(path.size() - 1); // backtrack

}

}

}

## Dry Run Using Table

**Input:** nums = [1, 2, 2]  
**Sorted:** [1, 2, 2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Path | Result So Far | i | Skipped? |
| 0 | [] | [[]] | - | - |
| 1 | [1] | [[], [1]] | 0 | No |
| 2 | [1, 2] | [[], [1], [1, 2]] | 1 | No |
| 3 | [1, 2, 2] | [[], [1], [1, 2], [1, 2, 2]] | 2 | No |
| ← | [1, 2] |  | ← | - |
| ← | [1] |  | ← | - |
| 4 | [1, 2] | Skipped (duplicate 2) | 2 | ✅ Yes |
| ← | [] |  | ← | - |
| 5 | [2] | [[], [1], [1, 2], [1, 2, 2], [2]] | 1 | No |
| 6 | [2, 2] | [... , [2, 2]] | 2 | No |
| ← | [2] |  | ← | - |
| ← | [] |  | ← | - |
| 7 | [2] | Skipped (duplicate 2) | 2 | ✅ Yes |

✅ **Final Result:**

[[], [1], [1,2], [1,2,2], [2], [2,2]]

## Time / Space Complexity

|  |  |  |
| --- | --- | --- |
| Metric | Value | Why |
| ⏱ Time | **O(2ⁿ)** | Generate all subsets, skipping duplicates |
| 💾 Space | **O(n)** | Max recursion stack + path length |

## Alternate Approaches

| **Approach** | **Description** |
| --- | --- |
| ✅ Sorting + Backtracking | Most standard, avoids duplicates by i > start check |
| ❌ Set + Brute Force | Can be used, but slower and unordered |
| ✅ Iterative (with queue) | Can build layer-by-layer while skipping duplicates |