# ****Leetcode 47 – Permutations II****

## Problem Understanding

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

### Constraints:

* Input can contain duplicates.
* The result **must not** contain duplicate permutations.

### Example:

Input: nums = [1,1,2]

Output: [[1,1,2], [1,2,1], [2,1,1]]

## Optimized Java Solution (Backtracking with Sorting + Used[] + Skip Duplicates)

class Solution {

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

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

Arrays.sort(nums); // Sort to handle duplicates

boolean[] used = new boolean[nums.length];

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

return result;

}

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

if (path.size() == nums.length) {

result.add(new ArrayList<>(path));

return;

}

for (int i = 0; i < nums.length; i++) {

// Skip used elements

if (used[i]) continue;

// Skip duplicates: if same as previous and previous was not used in current path

if (i > 0 && nums[i] == nums[i - 1] && !used[i - 1]) continue;

path.add(nums[i]);

used[i] = true;

backtrack(nums, used, path, result);

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

used[i] = false;

}

}

}

## Dry Run Using Table (nums = [1, 1, 2])

|  |  |  |  |
| --- | --- | --- | --- |
| Level | Path | Used | Action |
| 0 | [] | [F, F, F] | Start |
| 1 | [1] | [T, F, F] | Choose index 0 |
| 2 | [1,1] | [T, T, F] | Choose index 1 |
| 3 | [1,1,2] | [T, T, T] | ✅ Add result |
| back | [1,1] | [T, T, F] | Backtrack |
| back | [1] | [T, F, F] | Backtrack |
| 2 | [1,2] | [T, F, T] | Choose index 2 |
| 3 | [1,2,1] | [T, T, T] | ✅ Add result |
| back | [1,2] | [T, F, T] | Backtrack |
| back | [1] | [T, F, F] | Backtrack |
| back | [] | [F, F, F] | Backtrack |
| 1 | [2] | [F, F, T] | Choose index 2 |
| 2 | [2,1] | [T, F, T] | Choose index 0 |
| 3 | [2,1,1] | [T, T, T] | ✅ Add result |
| back |  |  |  |

✅ Final Output: [[1,1,2], [1,2,1], [2,1,1]]

## Time / Space Complexity

| **Metric** | **Value** |
| --- | --- |
| Time | O(n! \* n) in worst case |
| Space | O(n) recursion + O(n!) result |

* Sorting takes O(n log n)
* Total unique permutations ≤ n!
* For each, we use a path of length n

## Alternate Approaches

|  |  |  |
| --- | --- | --- |
| Approach | Time | Notes |
| ✅ Backtracking + used[] + skip dup | O(n!) | Most optimal and readable |
| In-place swapping with hash set | O(n!) | Avoids extra space but more complex |