# Title: Permutations II

## Overview:

This documentation provides an explanation of the problem "Permutations II" along with a solution approach. The problem requires generating unique permutations from a collection of numbers, where the collection might contain duplicates.

## Problem Description:

Given a collection of numbers represented by the list `nums`, which may contain duplicate elements, the task is to generate all possible unique permutations of the numbers in any order.

## Examples:

### Example 1:

Input: nums = [1,1,2]

Output:

[[1,1,2],

[1,2,1],

[2,1,1]]

### Example 2:

Input: nums = [1,2,3]

Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

## Constraints:

- 1 <= nums.length <= 8
- -10 <= nums[i] <= 10

## Solution Approach:

The solution utilizes backtracking to generate permutations while ensuring uniqueness. Here's a step-by-step breakdown of the solution approach:

1. **Sorting:** Sort the input list `nums`. Sorting the list is crucial to handle duplicate elements efficiently.

2. **Backtracking Function:**
    - Define a recursive backtracking function `backtrack(nums, path, res, used)` that takes four parameters:
    - `nums`: The input list of numbers.
    - `path`: A list representing the current permutation being constructed.
    - `res`: A list to store the resulting unique permutations.
    - `used`: A boolean list to keep track of used elements during permutation generation.
    - If the length of the `path` equals the length of `nums`, append a copy of `path` to `res` (this represents a valid permutation) and return.
    - Iterate through each element of `nums`:
    - If the element is already used or if it's a duplicate and its previous instance is unused, skip the current iteration.
    - Mark the current element as used.
    - Add the current element to the `path`.

- Recursively call `backtrack` with updated parameters.
- Mark the current element as unused (backtrack).
- Remove the last element from the `path`.

### 3. Main Function:

- Initialize an empty list `res` to store the resulting permutations.
- Initialize a boolean list `used` of length equal to the length of `nums`, where all elements are initially set to `False`.
- Call the `backtrack` function with initial parameters (`nums`, `[]`, `res`, `used`).
- Return the list `res` containing all unique permutations.

## Conclusion:

This documentation provides an understanding of the "Permutations II" problem, its constraints, and a detailed solution approach using backtracking. The solution efficiently handles duplicate elements to generate all unique permutations of the given collection of numbers.