```
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        permuteHelper(res, new ArrayList<>(), nums);
        return res;
    }

    private void permuteHelper(List<List<Integer>> res, List<Integer> tmp, int[] nums) {
        if (tmp.size() == nums.length) {
            res.add(new ArrayList<>(tmp));
            return;
        }
        for (int i : nums) {
            if (!tmp.contains(i)) {
                tmp.add(i);
                permuteHelper(res, tmp, nums);
                tmp.remove(tmp.size()-1);
            }
        }
    }
}
```

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]
]
```

```
class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();
        helper(nums, res, new ArrayList<>(), new boolean[nums.length]);
        return res;
    }

    private void helper(int[] nums, List<List<Integer>> res, List<Integer> cur, boolean[] used) {
        if (cur.size() == nums.length) {
            res.add(new ArrayList<>(cur));
            return;
        }

        for (int i = 0; i < nums.length; i++) {
            if (used[i] || (i > 0 && used[i-1] && !used[i] && nums[i-1] == nums[i])) continue;
            used[i] = true;
            cur.add(nums[i]);
            helper(nums, res, cur, used);
            cur.remove(cur.size()-1);
            used[i] = false;
        }
    }
}
```

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]
]
```