LeetCode 46. Permutations

1. Problem Title & Link

- **46. Permutations**
- 🔗 https://leetcode.com/problems/permutations/

2. Problem Statement (Short Summary)

Given an array nums of **distinct integers**, return *all possible permutations*.

A permutation is any possible **ordering** of the given elements.

3. Examples (Input → Output)

Input: nums = [1,2,3]

Output:

```
[
 [1,2,3],
 [1,3,2],
 [2,1,3],
 [2,3,1],
 [3,1,2],
 [3,2,1]
]
```

4. Constraints

- 1 <= nums.length <= 6
- -10 <= nums[i] <= 10
- All integers are unique.

5. Thought Process (Step by Step)

This is a **backtracking problem** that builds on combination/subset logic ❤️

The difference:

- In combinations, we only move *forward* (start+1).
- In permutations, we can choose *any element not yet used*.

🧠 Step 1: Recursive Choice

At each recursive call:

1. Loop through all elements in nums.
2. If an element hasn't been used in the current path:
    - Add it to the path.
    - Recurse to build further.
    - Backtrack (remove it again).

When the path length == len(nums) → we've built one valid permutation.

🧩 Step 2: Tracking Usage

Use a used[] boolean array (or a simple if num not in path check for smaller inputs).

6. Pseudocode

```
res = []

def backtrack(path):
    if len(path) == len(nums):
        res.append(copy(path))
        return

    for i in range(len(nums)):
        if nums[i] not in path:
            path.append(nums[i])
            backtrack(path)
            path.pop()
```

7. Code Implementation

✅ Python

```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = []

        def backtrack(path):
            if len(path) == len(nums):
                res.append(path[:])
                return
            for num in nums:
                if num not in path:
                    path.append(num)
                    backtrack(path)
                    path.pop()

        backtrack([])
        return res
```

✅ Java

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        boolean[] used = new boolean[nums.length];
        backtrack(nums, new ArrayList<>(), used, res);
        return res;
    }

    private void backtrack(int[] nums, List<Integer> path, boolean[] used,
List<List<Integer>> res) {
        if (path.size() == nums.length) {
            res.add(new ArrayList<>(path));
            return;
        }
        for (int i = 0; i < nums.length; i++) {
            if (used[i]) continue;
            used[i] = true;
            path.add(nums[i]);
            backtrack(nums, path, used, res);
            path.remove(path.size() - 1);
            used[i] = false;
        }
    }
}
```

8. Time & Space Complexity

Metric	Complexity	Reason
Time	O(n × n!)	Each permutation of length n generated.
Space	O(n)	Recursion stack and path storage.

9. Dry Run (Step-by-Step Execution)👉 Input: nums = [1,2,3]

Step	Path	Action	Result
1	[]	start	—
2	[1]	choose 1	—
3	[1,2]	choose 2	—
4	[1,2,3]	✅ store	[[1,2,3]]
5	[1,3]	choose 3	—
6	[1,3,2]	✅ store	[[1,2,3],[1,3,2]]
7	[2]	choose 2	—
8	[2,1,3]	✅ store	…
9	[2,3,1]	✅ store	…
10	[3,1,2]	✅ store	…

11	[3,2,1]	✅ store	Final ✅

✅ Output:

```
[
 [1,2,3],
 [1,3,2],
 [2,1,3],
 [2,3,1],
 [3,1,2],
 [3,2,1]
]
```

10. Visual Tree Diagram (For Class ❤️)

```
            []
     /        |         \
   [1]       [2]        [3]
  /  \      /  \       /  \
[1,2] [1,3]  [2,1] [2,3]  [3,1] [3,2]
  |     |     |     |     |     |
[1,2,3] [1,3,2] [2,1,3] [2,3,1] [3,1,2] [3,2,1]
```

Each **branch** = adding a new number

Each **leaf node** = a complete permutation 🌸

11. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Backtracking (Permutation Generation) — choose unused elements recursively to build orderings.	- Ordering problems - String rearrangements - Game move generation	- Forgetting to "pop" after recursion - Using same element multiple times - Modifying `path` directly without copy	🔷 Builds to LC 47 (Permutations II) 🔷 Connects to LC 51 (N-Queens) 🔷 Foundation for DFS search tree logic

12. Common Mistakes / Edge Cases

- Forgetting to backtrack (path.pop()).
- Using same element twice (missing used[] check).
- Not copying path (path[:] in Python).

13. Variations / Follow-Ups

- **LC 47:** Permutations II — handles duplicates.
- **LC 784:** Letter Case Permutation.
- **LC 60:** K-th Permutation Sequence.
- **LC 31:** Next Permutation (in-place reordering).