Documentation: Permutations

Problem Statement

Given an array `nums` of distinct integers, the task is to return all possible permutations of the array. The order of the permutations does not matter.

Examples

1. **Example 1:**
    - Input: `nums = [1,2,3]`
    - Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

2. **Example 2:**
    - Input: `nums = [0,1]`
    - Output: `[[0,1],[1,0]]`

3. **Example 3:**
    - Input: `nums = [1]`
    - Output: `[[1]]`

Constraints

- The length of `nums` is between 1 and 6 (inclusive).
- The integers in `nums` are between -10 and 10 (inclusive).
- All integers in `nums` are unique.

Solution Approach

The solution employs backtracking to generate all possible permutations of the given array. It swaps elements to generate permutations recursively, then backtracks by swapping elements back to their original positions.

Example usage:

solution = Solution()

print(solution.permute([1, 2, 3]))

print(solution.permute([0, 1]))

print(solution.permute([1]))

Explanation

- `permute()` function initializes the result list and calls the recursive permutation function.
- `permute_recursive()` function generates permutations recursively by swapping elements and backtracking. It appends a copy of the current permutation to the result list when the end of the array is reached.
- The solution follows the backtracking algorithm to generate all permutations of the given array.