# Question

Given an array nums of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

**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]]

**Example 2:**

**Input:** nums = [0,1]

**Output:** [[0,1],[1,0]]

**Example 3:**

**Input:** nums = [1]

**Output:** [[1]]

**Constraints:**

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

# Solution

#### **Approach 1: Backtracking**

[Backtracking](https://en.wikipedia.org/wiki/Backtracking) is an algorithm for finding all solutions by exploring all potential candidates. If the solution candidate turns to be not a solution (or at least not the last one), backtracking algorithm discards it by making some changes on the previous step, i.e. backtracks and then try again.

Here is a backtrack function which takes the index of the first integer to consider as an argument backtrack(first).

* If the first integer to consider has index n that means that the current permutation is done.
* Iterate over the integers from index first to index n - 1.
  + Place i-th integer first in the permutation, i.e. swap(nums[first], nums[i]).
  + Proceed to create all permutations which starts from i-th integer : backtrack(first + 1).
  + Now backtrack, i.e. swap(nums[first], nums[i]) back.

1 / 14

|  |
| --- |
| **class Solution {**  **public void backtrack(int n,**  **ArrayList<Integer> nums,**  **List<List<Integer>> output,**  **int first) {**  **// if all integers are used up**  **if (first == n)**  **output.add(new ArrayList<Integer>(nums));**  **for (int i = first; i < n; i++) {**  **// place i-th integer first**  **// in the current permutation**  **Collections.swap(nums, first, i);**  **// use next integers to complete the permutations**  **backtrack(n, nums, output, first + 1);**  **// backtrack**  **Collections.swap(nums, first, i);**  **}**  **}**  **public List<List<Integer>> permute(int[] nums) {**  **// init output list**  **List<List<Integer>> output = new LinkedList();**  **// convert nums into list since the output is a list of lists**  **ArrayList<Integer> nums\_lst = new ArrayList<Integer>();**  **for (int num : nums)**  **nums\_lst.add(num);**  **int n = nums.length;**  **backtrack(n, nums\_lst, output, 0);**  **return output;**  **}**  **}** |

**Complexity Analysis**

