Problem Statement - Permutations

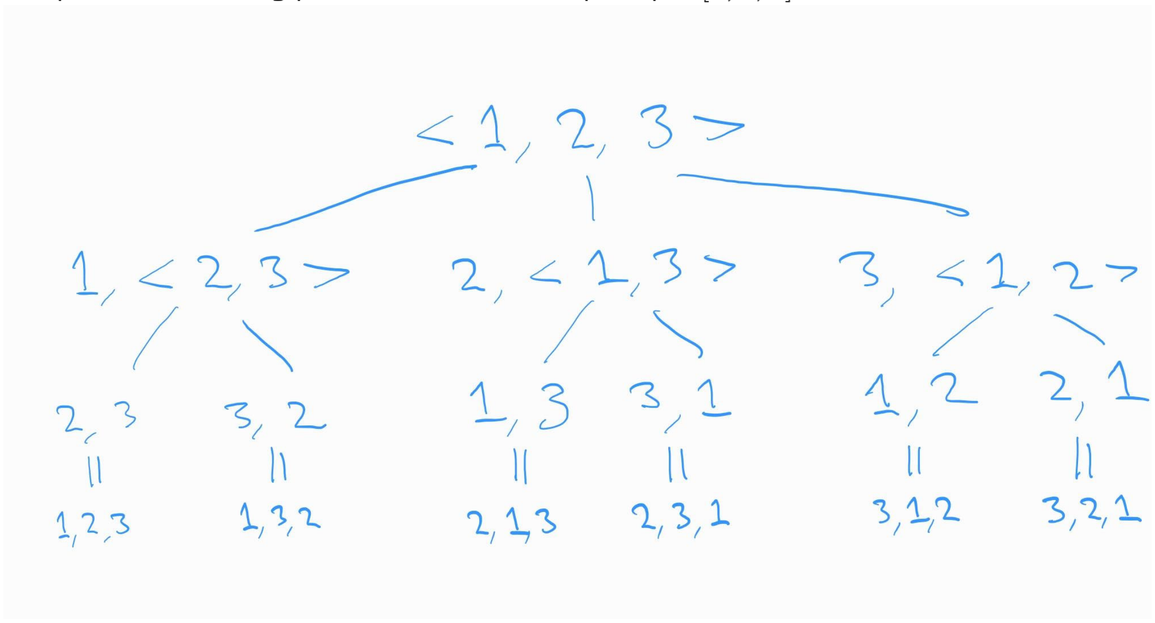
Given an array of numbers, find all permutations.

Assumptions

- The array can be converted into a list for ease of manipulation
- All elements are distinct

Explanation of Recursive Solution

The process for finding permutations for a sample input $[1, 2, 3]$ is as follows:



Notice how there are angle brackets present in every non-leaf node. These represent smaller subarrays for which all permutations are solutions. That is, to find permutations, you must find more permutations. But to find those permutations, you must find yet again more permutations. Initially, this cycle of self-reference appears paradoxical.

But fear not! The crucial detail is that the size of the subarrays progressively decreases with each self-reference. And once the subarrays reach size 1, the problem becomes trivial enough to be solved without further self-reference, resolving the paradox. There is only 1 permutation of a size 1 array, itself.

So this can be solved using a function that repeatedly calls itself and performs calculations until a base case is reached in a process known as recursion. After the results of base case function calls have been computed, the function instance that called them prepends a fixed number to obtain 2 permutations for a subproblem. The process repeats up the function call hierarchy until the original problem of finding all permutations is solved.

Pseudocode

```
define $Permute(nums)$ :
|   $len \leftarrow$ length of $nums$

|   //base case
|   if $len == 1$ :
|   |   return a list containing $nums$
|   endif

|   $perms \leftarrow$ empty list
|   for every $i \in \mathbb{N}$ from 1 to $len$ :
|   |   $fixed \leftarrow i$th element of $nums$
|   |   $remain \leftarrow nums$ with $i$th element removed

|   |   //recursive calls
|   |   for each element $sub$ in list returned from $Permute(remain)$ :
|   |   |   $perm \leftarrow sub$ with $fixed$ prepended
|   |   |   append $perm$ to $perms$
|   |   return $perms$


input $num$
convert $num$ to a list
output $Permute(nums)$
```

Python Code for Demonstration

In [ ]:
```
def permute(nums):
    l = len(nums)

    # base case
    if l == 1:
      return [nums]

    perms = []
    for i in range(l):
      fixed = nums[i]
      remain = nums[:i] + nums[i+1:]

      # recursive calls
      for sub in permute(remain):
        perms.append([fixed] + sub)
    return perms

n = int(input("Size of array: "))
nums = []
for i in range(n):
   nums.append(int(input(f"Enter element {i+1}: ")))

print(permute(nums))
```