Print All Permutations of a String/Array

Problem Statement: Given an array arr of distinct integers, print all permutations of String/Array.

Examples:

```
Example 1:

Input: arr = [1, 2, 3]

Output:
[
  [1, 2, 3],
  [1, 3, 2],
  [2, 1, 3],
  [2, 3, 1],
  [3, 1, 2],
  [3, 2, 1]
]

Explanation: Given an array, return all the possible permutations.

Example 2:

Input: arr = [0, 1]
```

Output:

```
[

  [0, 1],

  [1, 0]

]
```

Explanation: Given an array, return all the possible permutations.

Solution

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Recursive

Approach: We have given the nums array, so we will declare an ans vector of vector that will store all the permutations also declare a data structure.

Declare a map and initialize it to zero and call the recursive function

Base condition:

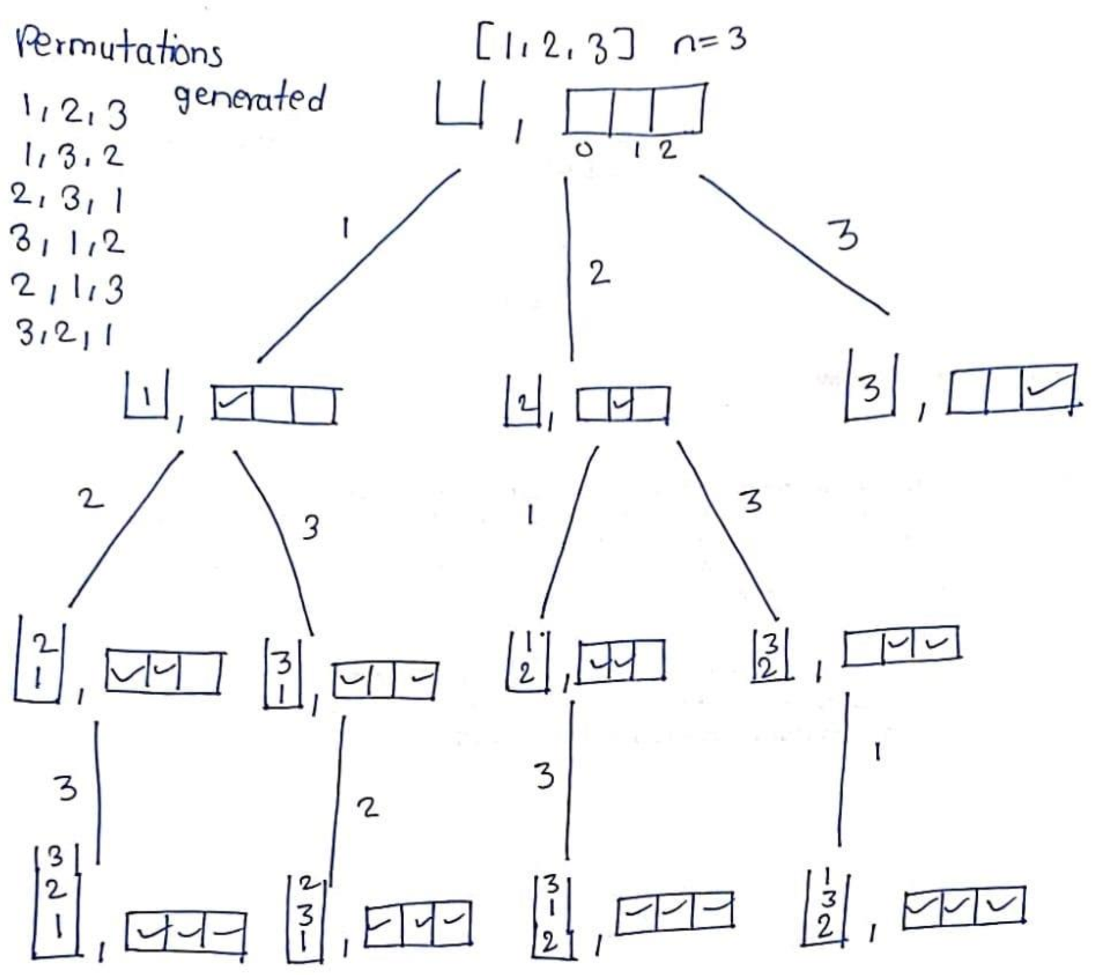
When the data structure's size is equal to n(size of nums array) then it is a permutation and stores that permutation in our ans, then returns it.

Recursion:

Run a for loop starting from 0 to nums.size() – 1. Check if the frequency of i is unmarked, if it is unmarked then it means it has not been picked and then we pick. And make sure it is marked as picked.

Call the recursion with the parameters to pick the other elements when we come back from the recursion make sure you throw that element out. And unmark that element in the map.

Recursive Tree:



Code:

```
class Solution {

    public List<List<Integer>> permute(int[] nums) {

        List<List<Integer>> res = new ArrayList();

        boolean[] taken = new boolean[nums.length];

        helper(nums,taken,res, new ArrayList<Integer> ());

        return res;

    }

    void helper(int[] nums,boolean[] taken,List<List<Integer>> res,
List<Integer> ds)

    {
        if(ds.size()==nums.length)
        {
            res.add(new ArrayList(ds));
            return;
        }
        for(int i=0;i<nums.length;i++)
        {
            if(taken[i])
                continue;
            ds.add(nums[i]);
            taken[i]=true;
            helper(nums,taken,res,ds);
            ds.remove(ds.size()-1);
            taken[i]=false;
        }
    }
}
```

Time Complexity: N! x N

Space Complexity: O(N)

Note: this approach is used to generate permutations in lexicographical order.

Solution 2: With Backtracking.

Approach: Using backtracking to solve this.

We have given the nums array, so we will declare an ans vector of vector that will store all the permutations.

Call a recursive function that starts with zero, nums array, and ans vector.

Declare a map and initialize it to zero and call the recursive function

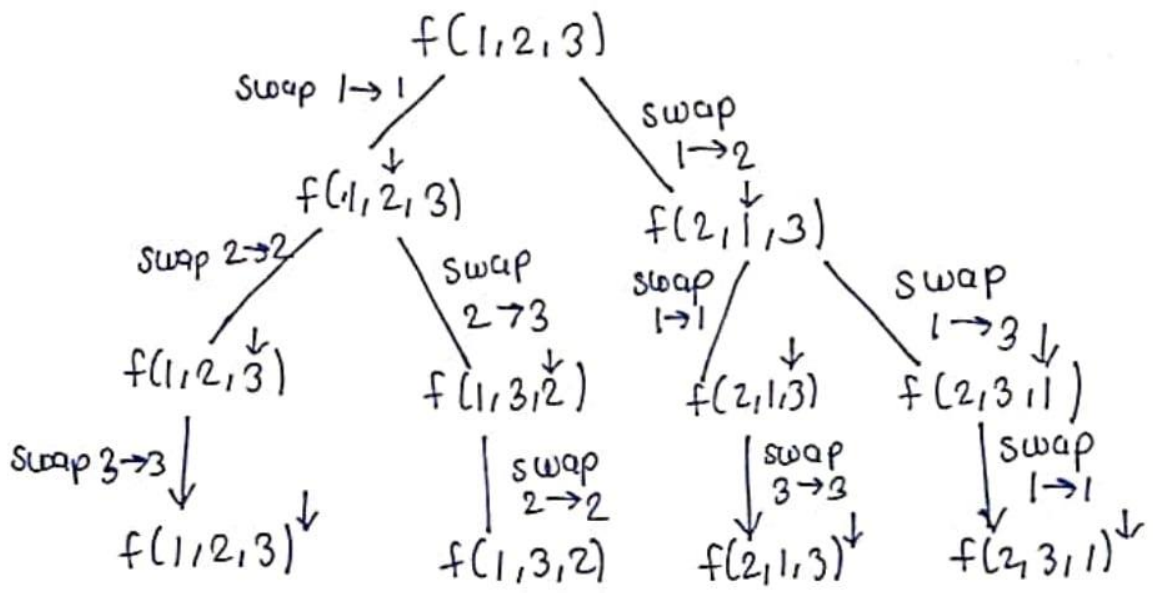
Base condition:

Whenever the index reaches the end take the nums array and put it in ans vector and return.

Recursion:

Go from index to n – 1 and swap. Once the swap has been done call recursion for the next state.After coming back from the recursion make sure you re-swap it because for the next element the swap will not take place.

Recursive Tree:



Code:

```
class Solution {
```

```
public List<String> find_permutation(String S) {

   List<String> res=new <String>();

   helper(new StringBuilder(S),res,0);

   return res;

}

public void helper(StringBuilder s, List<String> res,int ind)

{

    if(ind>=s.length())

    {

        res.add(s.toString());

        return;

    }

    for(int i=ind;i<s.length();i++)

    {

        swap(s,i,ind);

        helper(s,res,ind+1);

        swap(s,i,ind);

    }

}

public void swap(StringBuilder s,int i,int j)

{

    char c=s.charAt(i);

    s.setCharAt(i,s.charAt(j));
```

```
        s.setCharAt(j,c);
    }
}
```

Time Complexity: O(N! X N)

Space Complexity: O(1)