12. LC 46 Print all Permutations of a String/Array | Recursion | Approach - 1

46. Permutations

Medium 11464 204 Add to List Share

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:

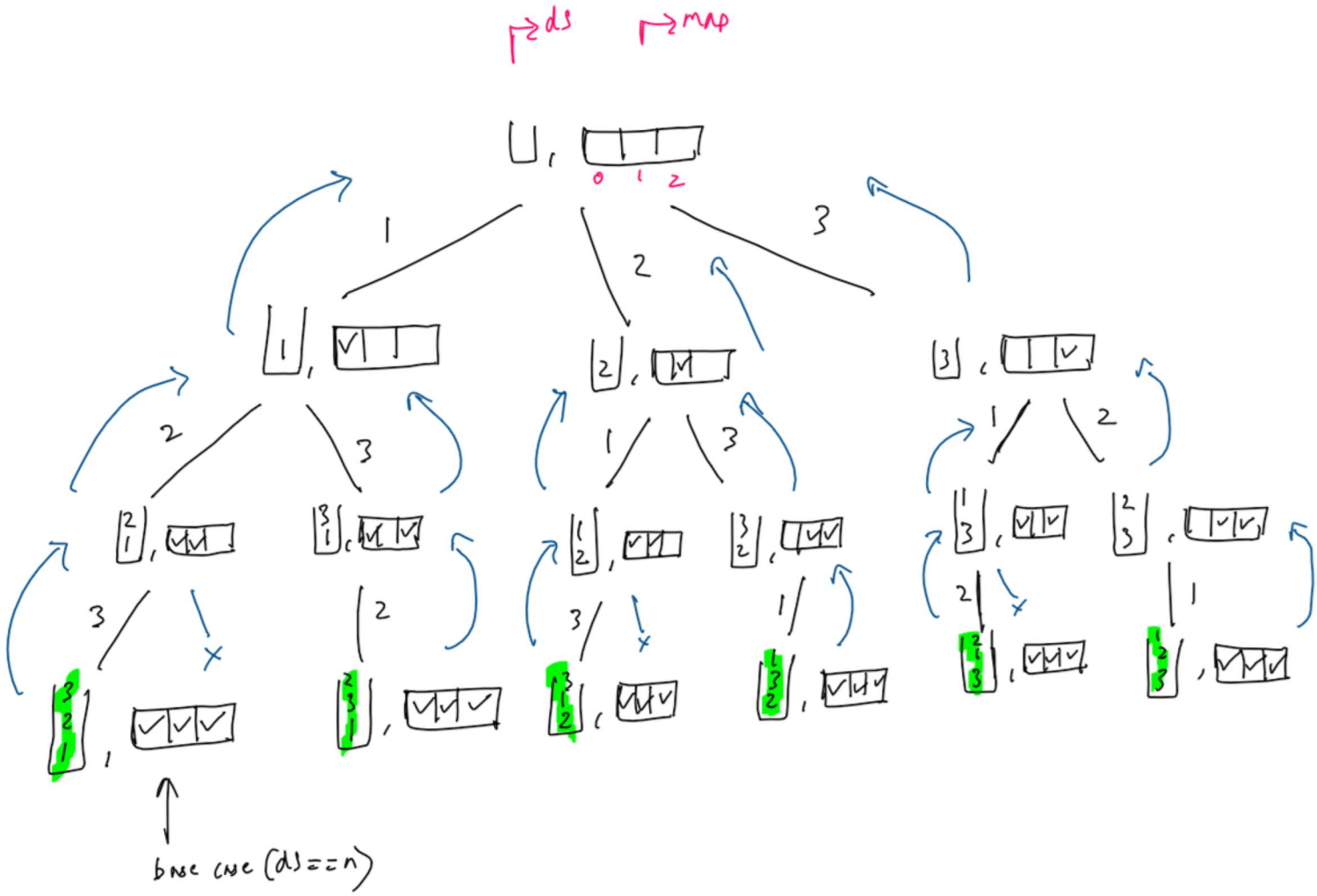
```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Approach-1 : with extra space complexility.

[1, 2, 3]

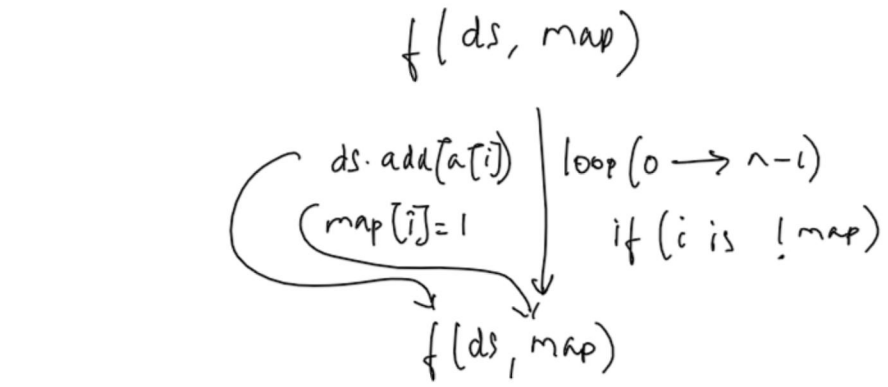
n = 3

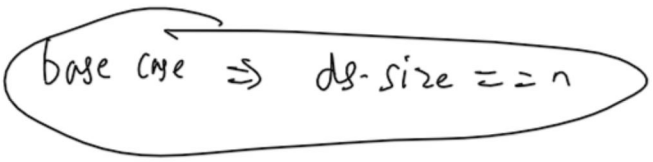
n! = 3! = 6



O/P ⇒ 1 2 3 , 1 3 2 , 2 1 3 , 2 3 1 , 3 1 2 , 3 2 1 // Permutation generated.

2 2 2





T.C ⇒ O(n! n) S.C ⇒ O(n) + O(n) ← map.

Steps :

* 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 recursion functions.
* Base Condition ⇒ ds.size == n, then it's a permutation and store that it our ans, then return it.
* Recursive Case ⇒

  → For loop 0 to num.size() - 1, check if the frequency

→ For loop 0 to num.size() - 1, check if the frequency of 'i' is unmarked, if it's unmarked then it means it has not been picked and then we pick, and make sure it's marked as picked.

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

Java · Autocomplete

```java
// Approach - 1
class Solution {
    private void rPermute(int[] nums, List<Integer> ds, List<List<Integer>> ans, boolean [] freq){
        // base case
        if(ds.size() == nums.length){
            ans.add(new ArrayList<>(ds));
            return;
        }
        for(int i = 0; i < nums.length; i++){
            if(!freq[i]){
                freq[i] = true;
                ds.add(nums[i]);
                rPermute(nums, ds, ans, freq);
                ds.remove(ds.size() - 1);
                freq[i] = false;
            }
        }
    }

    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> ds = new ArrayList<>();
        boolean freq[] = new boolean[nums.length]; // frequency array of same size as nums
        rPermute(nums, ds, ans, freq);
        return ans;
    }
}
```

C++ Autocomplete

```cpp
// Approach - 1
class Solution {
private:
    void rPermute(vector<int> &ds, vector<int> &nums, vector<vector<int>> &ans, int freq[]){
        if(ds.size() == nums.size()){
            ans.push_back(ds);
            return;
        }
        for(int i = 0; i < nums.size(); i++){
            if(!freq[i]){
                ds.push_back(nums[i]);
                freq[i] = 1;
                rPermute(ds, nums, ans, freq);
                freq[i] = 0;
                ds.pop_back();
            }
        }
    }
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> ds;
        int freq[nums.size()];
        for(int i = 0; i < nums.size(); i++) freq[i] = 0;
        rPermute(ds, nums, ans, freq);
        return ans;
    }
};
```