17. Permutations

Permutations

A permutation is a rearrangement of members of a sequence into a new sequence.

Problem:

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

Sample test case

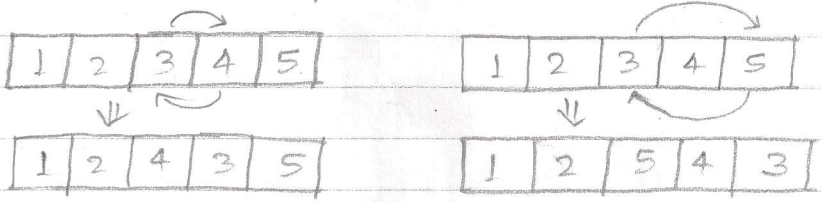
n!

Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2] [3,2,1]]

Input: [0,1]
Output: [[0,1],[1,0]]

IDEA

You are at index idx, try out all the possible swap Ensuring you don't produce duplicates.



Pseudocode

```
if (idx == nums.size()) {
    ans.push_back(nums);
    return;
}
for(int i=idx; i<nums.size(); i++){
    swap(nums[i], nums[idx]);
    solve(nums, ans, idx+1);
    swap(nums[i], nums[idx]);
}
```

Complexity

Permutations of N distinct elements, so any algorithm to generate permutations - even using recursion - will be very slow to finish (at least, O(N!)) when N is large.

Time Complexity: O(N!)

Code

```
vector<vector<int>> ans;

void permute(vector<int> &a, int idx){
    if (idx == a.size()) {
        ans.push_back(a);
        return;
    }
    for(int i=idx; i<a.size(); i++) {
        swap(a[i], a[idx]);
        permute(a, idx+1);
        swap(a[i], a[idx]);
    }
}
```

Usage:

```
vector<int> a(n);
permute(a, 0);
```

STL Trick !

```
sort (nums.begin() , nums.end());
do {
    ans.push_back (nums);
} while (next_permutation (nums.begin(), nums.end()));
```

Time Complexity : O(N!)

Code

```
#include <algorithm>   // for next_permutation & sort function

int main () {
    int n ; cin >>n ;
    vector <int> a(n);
    for (auto &i : a)
        cin >>i ;
    vector <vector <int>> ans ;

    sort (a.begin() , a.end());
    do {
        ans. push_back (a);
    } while (next_permutation (a.begin() , a.end()));

    for (auto v: ans) {
        for (auto i :v)
            cout << i << " ";
        cout << "\n";
    }
    return 0;
}
```

Example: 3 1 2 3

o/p ⇒ 1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Permutation - II

Problem

Given a collection of numbers, nums, that might contain duplicates, return all possible unique permutations in any order.

Avoid Duplicates

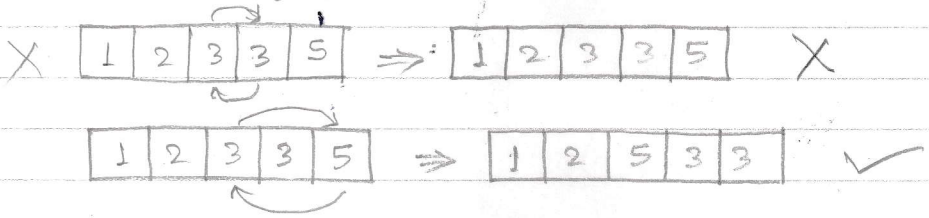
Sample Test Case

Input : nums = [1,1,2]
Output : [[1,1,2], [1,2,1], [2,1,1]]

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

Idea

While swapping, Avoid Duplicates



Pseudocode

```
if (idx == a.size()) {
    ans.push_back(a);
    return;
}
for (int i = idx; i < a.size(); i++) {
    if ( i != idx and  a[i] == a[idx])
        continue;
    swap (a[i], a[idx]);
    helper (a, idx+1);
}
```

Code:

```
#include <algorithm>   // for sort function

void helper (vector <int> a, vector <vector <int>> &ans, int idx) {
        if (idx == a.size()) {
            ans.push_back(a);
            return;
        }
        for (int i = idx; i < a.size (); i++) {
            if (idx != i and a[idx] == a[i])
                continue;
            swap (a[i], a[idx]);
            helper (a, ans, idx+1);
        }
}

vector <vector < int>> permute (vector <int> a) {
        sort (a.begin(), a.end());
        vector <vector <int>> ans;
        helper (a, ans, 0);
        return ans;
}

int main () {
    int n; cin >> n;
    vector <int> a(n);
    for (auto &i : a)
        cin >> i;
    vector < vector <int>> res = permute (a);
    for (auto v : res) {
        for (auto i : v)
            cout << i << " ";
        cout << "\n";
    }
    return 0;
}
```

input: 3
1 2 2

output: 1 2 2
2 1 2
2 2 1