46. Permutations

This is a classic backtracking / complete search problem. The general approach to backtracking is to check the condition, append new element, check the condition, then remove new element.

The method is best understood through the code itself.

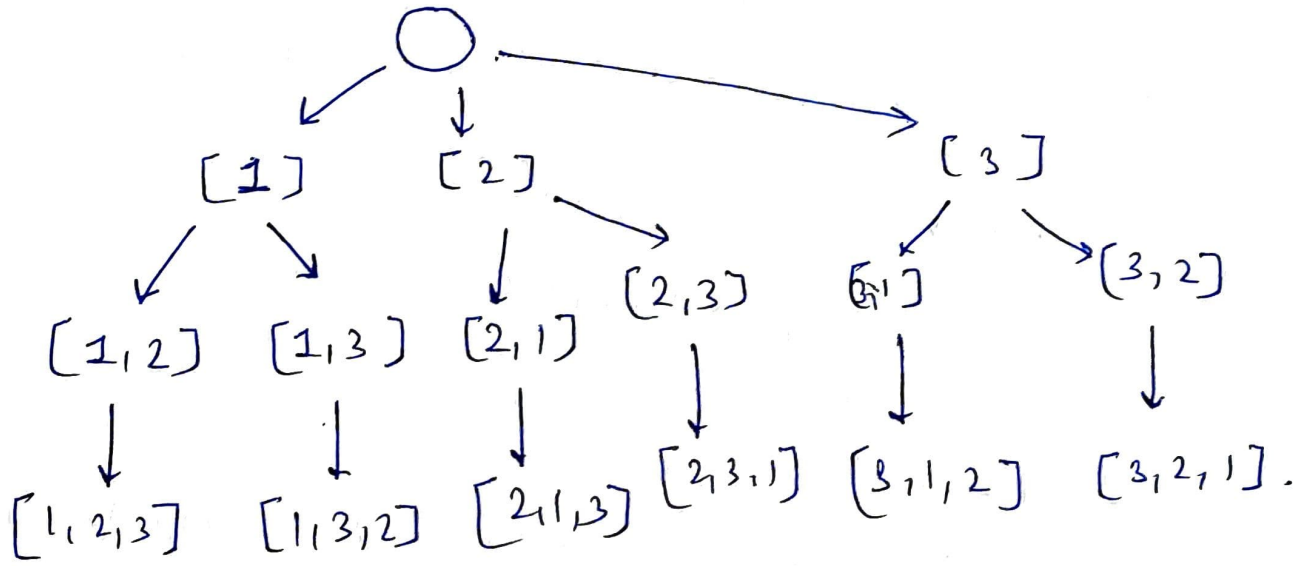
CODE

```
vector < vector < int > > v;
void backtrack ( vector <int> nums, veetor <int> v1, int n ){
        if ( v1. size() == n){
                v. push_back(v1);
                retuen ;
            }
    for ( int i = 0 ; i < nums. size() ; i++) {
1               v1. push_back ( nums [i] );
2               vector <int> ne = nums;
3               ne. erase ( find(ne. begin(), ne. end(), ne[i]));
4               backtrack ( ne, v1, n);
5               v1. pop_back ();
6       }.
```

→

```
vector <vector <int>> permute (vector <int> & nums){ (2)
        int n = nums.size();
        vector <int> v1;
        backtrack ( nums, v1, n);
        return v;
}.
```

→ Flow of the problem (left to right)



Time complexity → For this case, time complexity is equal to number of nodes in the above tree like structure. Last level has $n!$ nodes.

$$T(n) = 1 + n + n(n-1) + n(n-1)(n-2) \cdots n!$$

$$= n! \left(\frac{1}{n!} + \frac{1}{(n-1)!} + \frac{1}{(n-2)!} \cdots 1 \right)$$

$$T(n) = n! \left(\frac{1}{1} + \frac{1}{2!} \cdots \frac{1}{n!} \right) \leq n! \left(1 + \frac{1}{2} + \frac{1}{2^2} \cdots \frac{1}{2^n} \right)^{(2)}$$

$$\leq n! \cdot 2$$

$$\lim_{n \to \infty} T(n)$$

Thus $\underset{n \to \infty}{T(n)} = \underline{O(n!)}$

Now, we also have a cross function that uses find. find is $O(n)$

Thus Time complexity $\rightarrow O(n \cdot n!)$

Space complexity $\rightarrow O(n)$