Permutations

Given a collection of **distinct** numbers, return all possible permutations.

For example,
`[1,2,3]` have the following permutations:
`[1,2,3]`, `[1,3,2]`, `[2,1,3]`, `[2,3,1]`, `[3,1,2]`, and `[3,2,1]`.

Solution 1

This recursive solution is the my first response for this problem. I was surprised when I found no similar solution posted here. It is much easier to understand than DFS-based ones, at least in my opinion. Please find more explanations here. All comments are welcome.

```
class Solution {
public:
    vector<vector<int> > permute(vector<int> &num) {
        vector<vector<int> > result;

        permuteRecursive(num, 0, result);
        return result;
    }

    // permute num[begin..end]
    // invariant: num[0..begin-1] have been fixed/permuted
    void permuteRecursive(vector<int> &num, int begin, vector<vector<int> > &resu
lt)    {
        if (begin >= num.size()) {
            // one permutation instance
            result.push_back(num);
            return;
        }

        for (int i = begin; i < num.size(); i++) {
            swap(num[begin], num[i]);
            permuteRecursive(num, begin + 1, result);
            // reset
            swap(num[begin], num[i]);
        }
    }
};
```

written by xiaohui7 original link here

Solution 2

the basic idea is, to permute n numbers, we can add the nth number into the resulting `List<List<Integer>>` from the n-1 numbers, in every possible position.

For example, if the input num[] is {1,2,3}: First, add 1 into the initial `List<List<Integer>>` (let's call it "answer").

Then, 2 can be added in front or after 1. So we have to copy the List in answer (it's just {1}), add 2 in position 0 of {1}, then copy the original {1} again, and add 2 in position 1. Now we have an answer of {{2,1},{1,2}}. There are 2 lists in the current answer.

Then we have to add 3. first copy {2,1} and {1,2}, add 3 in position 0; then copy {2,1} and {1,2}, and add 3 into position 1, then do the same thing for position 3. Finally we have 2*3=6 lists in answer, which is what we want.

```
public List<List<Integer>> permute(int[] num) {
    List<List<Integer>> ans = new ArrayList<List<Integer>>();
    if (num.length ==0) return ans;
    List<Integer> l0 = new ArrayList<Integer>();
    l0.add(num[0]);
    ans.add(l0);
    for (int i = 1; i< num.length; ++i){
        List<List<Integer>> new_ans = new ArrayList<List<Integer>>();
        for (int j = 0; j<=i; ++j){
           for (List<Integer> l : ans){
               List<Integer> new_l = new ArrayList<Integer>(l);
               new_l.add(j,num[i]);
               new_ans.add(new_l);
           }
        }
        ans = new_ans;
    }
    return ans;
}
```

---

python version is more concise:

```
def permute(self, nums):
    perms = [[]]
    for n in nums:
        new_perms = []
        for perm in perms:
            for i in xrange(len(perm)+1):
                new_perms.append(perm[:i] + [n] + perm[i:])   ###insert n
        perms = new_perms
    return perms
```

written by cbmbbz original link here

Solution 3

```
public List<List<Integer>> permute(int[] num) {
    LinkedList<List<Integer>> res = new LinkedList<List<Integer>>();
    res.add(new ArrayList<Integer>());
    for (int n : num) {
        int size = res.size();
        for (; size > 0; size--) {
            List<Integer> r = res.pollFirst();
            for (int i = 0; i <= r.size(); i++) {
                List<Integer> t = new ArrayList<Integer>(r);
                t.add(i, n);
                res.add(t);
            }
        }
    }
    return res;
}
```

written by tusizi original link here