46. Permutations (/problems/permutations/)

Jan. 20, 2019 | 33.3K views

Average Rating: 4.37 (27 votes)

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

Example:

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

Solution

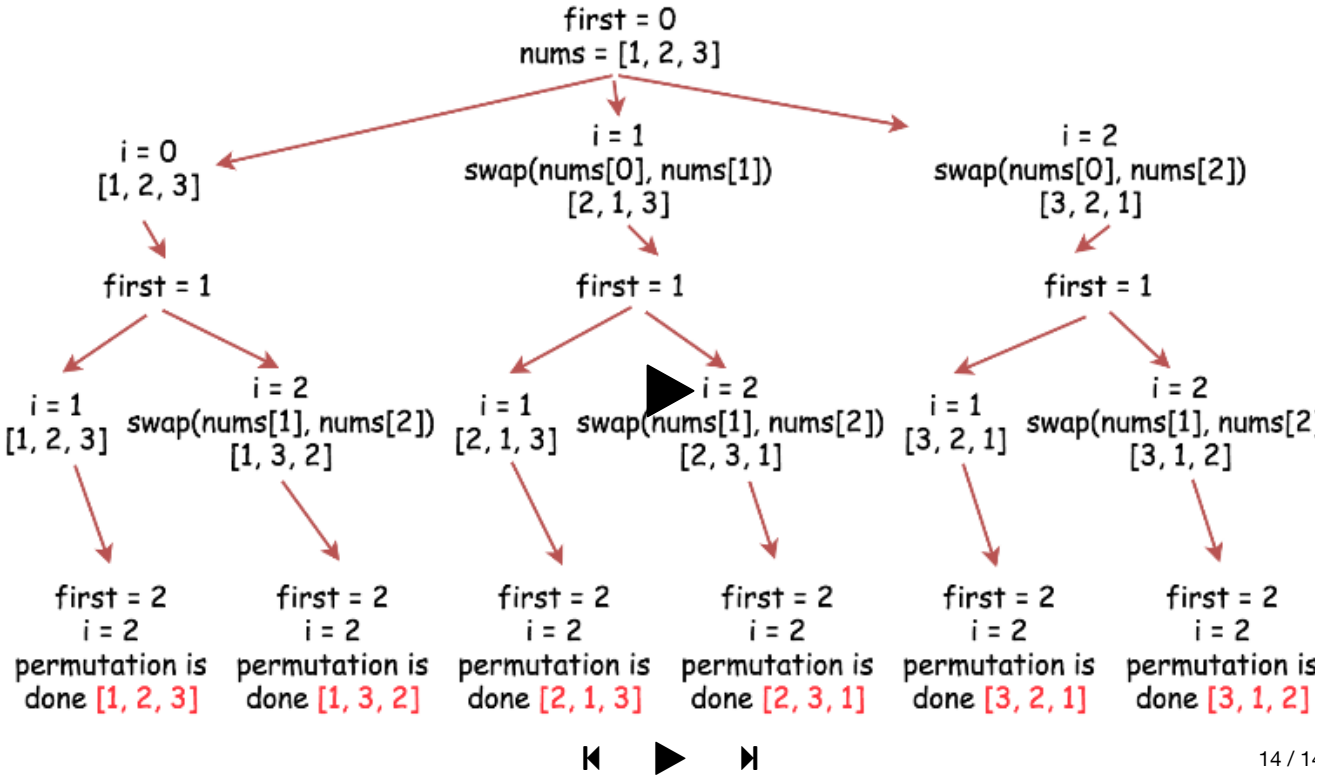
Approach 1: Backtracking

Backtracking (https://en.wikipedia.org/wiki/Backtracking) is an algorithm for finding all solutions by exploring all potential candidates. If the solution candidate turns to be *not* a solution (or at least not the *last* one), backtracking algorithm discards it by making some changes on the previous step, *i.e.* *backtracks* and then try again.

Here is a backtrack function which takes the index of the first integer to consider as an argument `backtrack(first)`.

- If the first integer to consider has index `n` that means that the current permutation is done.
- Iterate over the integers from index `first` to index `n - 1`.

- Place `i`-th integer first in the permutation, i.e. `swap(nums[first], nums[i])`.
- Proceed to create all permutations which starts from `i`-th integer : `backtrack(first + 1)`.
- Now backtrack, i.e. `swap(nums[first], nums[i])` back.



Java | Python

Articles > 46. Permutations

Copy

```python
class Solution:
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        def backtrack(first = 0):
            # if all integers are used up
            if first == n:
                output.append(nums[:])
            for i in range(first, n):
                # place i-th integer first
                # in the current permutation
                nums[first], nums[i] = nums[i], nums[first]
                # use next integers to complete the permutations
                backtrack(first + 1)
                # backtrack
                nums[first], nums[i] = nums[i], nums[first]

        n = len(nums)
        output = []
        backtrack()
        return output
```

Complexity Analysis

- Time complexity : $\mathcal{O}(\sum_{k=1}^{N} P(N, k))$ where $P(N, k) = \frac{N!}{(N-k)!} = N(N-1)...(N-k+1)$ is so-called *k-permutations_of_n*, or *partial permutation* (https://en.wikipedia.org/wiki/Permutation#k-permutations_of_n).

Here $first + 1 = k$ for the expression simplicity. The formula is easy to understand : for each k (each $first$) one performs $N(N-1)...(N-k+1)$ operations, and k is going through the range of values from 1 to N (and $first$ from 0 to $N-1$).

Let's do a rough estimation of the result : $N! \leq \sum_{k=1}^{N} \frac{N!}{(N-k)!} = \sum_{k=1}^{N} P(N, k) \leq N \times N!$, i.e. the algorithm performs better than $\mathcal{O}(N \times N!)$ and a bit slower than $\mathcal{O}(N!)$.

- Space complexity : $\mathcal{O}(N!)$ since one has to keep `N!` solutions.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/squares-of-a-sorted-array/) Next (/articles/restore-ip-addresses/)

Comments: 11

Articles > 46. Permutations

Sort By

Type comment here... (Markdown is supported)

Preview Post

(/s961206)

s961206 (s961206) ★ 316 January 28, 2019 2:42 AM

I think the time complexity is O(n x n!) instead of O(n!), since you will have n! permutation. And, for each permutation, you run exact n recursive call to reach it. So it should be n x n! ?

20 Share Reply

SHOW 1 REPLY

(/qqwei)

qqwei (qqwei) ★ 28 May 7, 2019 12:41 PM

For the complexity, I think you can explain in this way: in the first level of the tree, you have N options and for each of the option, you have N-1 option, and for each of these N-1 options, you have another N-2 options, so putting them together you would end up N*(N-1)*(N-2).... = N!

8 Share Reply

(/bobxu1128)

bobxu1128 (bobxu1128) ★ 93 February 6, 2019 1:43 PM

why we need the second swap?

8 Share Reply

SHOW 3 REPLIES

(/calvinchankf)

calvinchankf (calvinchankf) ★ 1122 April 11, 2019 4:27 AM

honestly, i dont really know how to analyze the time complexity of this approach. I understand the nPk part, but how can to come up with the summation and how can i present it to the interviewer?

6 Share Reply

(/alphaorc)

alphaorc (alphaorc) ★ 7 June 10, 2019 9:15 AM

For those interested, this is called Heap's Algorithm.

6 Share Reply

(/i_am_an_engineer)

i_am_an_engineer (i_am_an_engineer) ★ 1 July 11, 2019 3:41 PM

I wonder which one is the better implementation: Writing the recursion function `backtrack` inside the main function `permute` ? Or writing `backtrack` beside `permute` ?

Any comment is appreciated.

0 Share Reply

(/codemani)

codemani (codemani) ★ 0 May 27, 2019 6:48 AM

Why is the permutation not in order using this solution?

0 Share Reply

(/douer233)

douer233 (douer233) ★ 9 May 8, 2019 1:29 AM

Anyone can explain why the there should be output.append(nums[:]) but not nums?

0 Share Reply

SHOW 1 REPLY

(/harsh161)

harsh161 (harsh161) ★ 3 March 27, 2019 8:28 AM

Hello,
Can anyone explain why space complexity is O(N!) (factorial here) ? Shouldn't it be O(N) only (the depth of recursive tree) for Java program at least ?

My thinking here: At every index i, we go down the depth level from (i+1) to N and recurse

Read More

0 Share Reply

SHOW 4 REPLIES

(/hk10nis)

hk10nis (hk10nis) ★ 10 March 11, 2019 9:58 PM

Have you considered the slice operation for the time complexity?
I think the time complexity should be O(n * n!).

0 Share Reply