46. Permutations

Jan. 20, 2019 | 96.3K views

Average Rating: 4.08 (53 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]
]
```

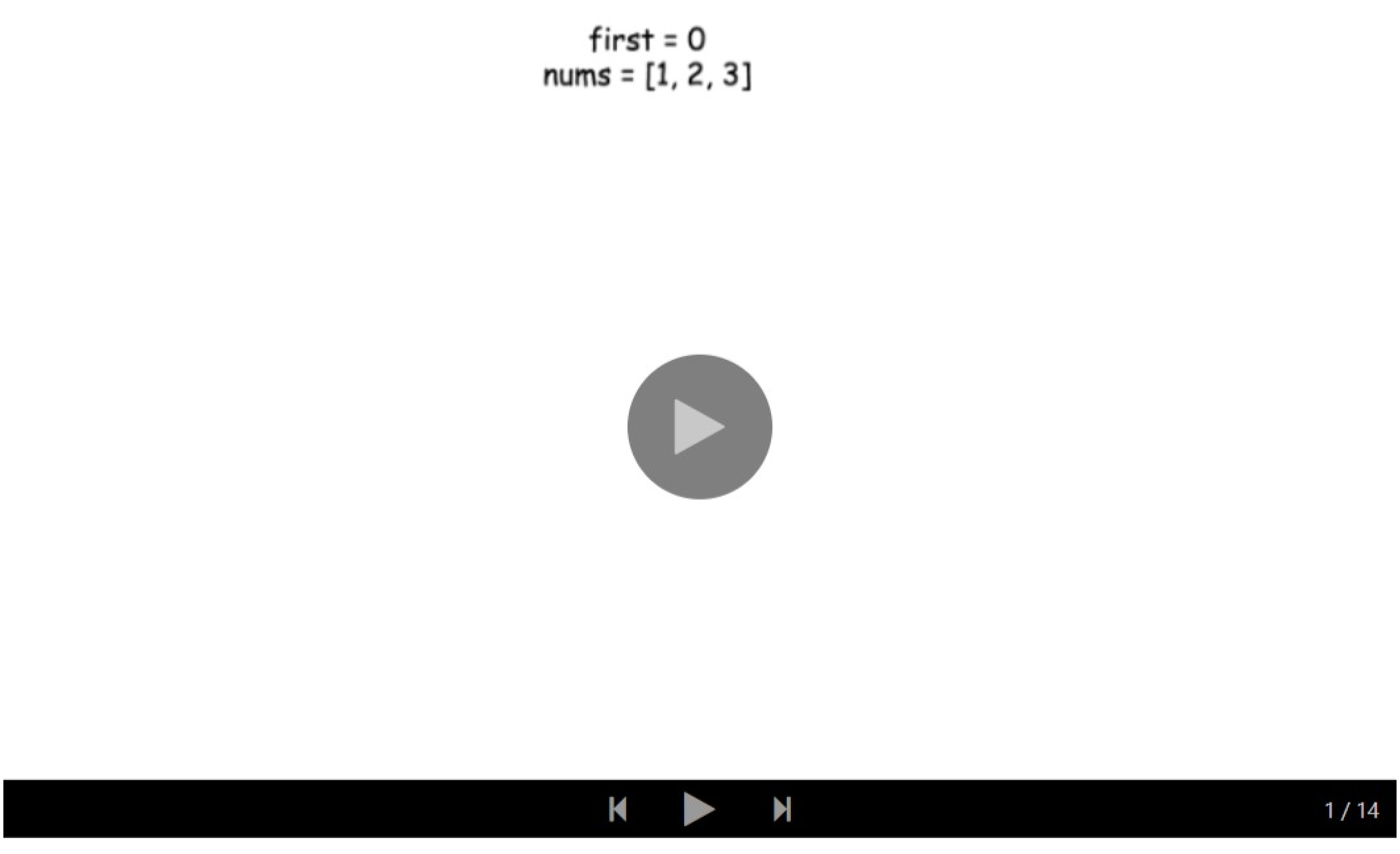
Solution

Approach 1: Backtracking

Backtracking 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

```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*.

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 : $N! \le \sum_{k=1}^{N} \frac{N!}{(N-k)!} = \sum_{k=1}^{N} P(N,k) \le N \times N!$, 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.

Rate this article:

Previous Next

Comments: 27

Sort By

Type comment here... (Markdown is supported)

Preview Post

s961206 ★ 733 January 28, 2019 4:12 PM

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! ?

60 Share Reply

SHOW 10 REPLIES

qqwei ★ 76 May 8, 2019 1:11 AM Report

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!

37 Share Reply

SHOW 3 REPLIES

junhaowanggg ★ 534 September 21, 2019 11:49 AM

First, I think the time complexity should be `N x N!`.

Initially we have `N` choices, and in each choice we have `(N - 1)` choices, and so on. Notice that at the end when adding the list to the result list, it takes `O(N)`.

Read More

21 Share Reply

calvinchankf ★ 2917 April 11, 2019 4:57 PM Report

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?

19 Share Reply

SHOW 2 REPLIES

alphaorc ★ 29 June 10, 2019 9:45 PM

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

27 Share Reply

bobxu1128 ★ 187 February 7, 2019 3:13 AM

why we need the second swap?

12 Share Reply

SHOW 4 REPLIES

douer233 ★ 32 May 8, 2019 1:59 PM Report

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

5 Share Reply

SHOW 1 REPLY

harsh161 ★ 9 March 27, 2019 8:58 PM

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 back, pop

Read More

3 Share Reply

SHOW 4 REPLIES

NeosDeus ★ 268 January 15, 2020 12:55 AM Report

Thought that the space required to store the final answer is usually not counted for space complexity calculation? Space should be O(N) due to search recursion stack space occupied by the DFS. As for people to suggested the time be N*N! for , I don't think having an extra N would change the Asymtotic complexity of N!

2 Share Reply

SHOW 1 REPLY

hk10nis ★ 23 March 12, 2019 10:28 AM Report

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

2 Share Reply

SHOW 1 REPLY

‹ 1 2 3 ›