

Permutations (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity
- Recursive Solution

Problem Statement

Given a set of distinct numbers, find all of its permutations.

Permutation is defined as the re-arranging of the elements of the set. For example, {1, 2, 3} has the following six permutations:

1. {1, 2, 3}
2. {1, 3, 2}
3. {2, 1, 3}
4. {2, 3, 1}
5. {3, 1, 2}
6. {3, 2, 1}

If a set has 'n' distinct elements it will have $n!$ permutations.

Example 1:

```
Input: [1,3,5]
Output: [1,3,5], [1,5,3], [3,1,5], [3,5,1], [5,1,3], [5,3,1]
```

Try it yourself

Try solving this question here:

Java

Python3

JS JS

C++

```
1 import java.util.*;
2
3 class Permutations {
4
5     public static List<List<Integer>> findPermutations(int[] nums) {
6         List<List<Integer>> result = new ArrayList<>();
7         // TODO: Write your code here
8         return result;
9     }
10
11     public static void main(String[] args) {
12         List<List<Integer>> result = Permutations.findPermutations(new int[] { 1, 3, 5 });
13         System.out.print("Here are all the permutations: " + result);
14     }
15 }
16
```

Run

Save

Reset

Solution

This problem follows the [Subsets](#) pattern and we can follow a similar **Breadth First Search (BFS)** approach. However, unlike [Subsets](#), every permutation must contain all the numbers.

Let's take the example-1 mentioned above to generate all the permutations. Following a BFS approach, we will consider one number at a time:

1. If the given set is empty then we have only an empty permutation set: []

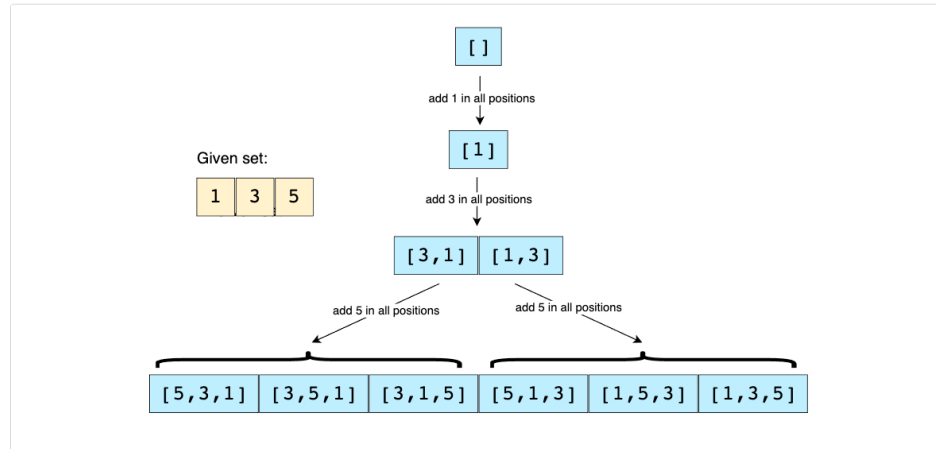
2. Let's add the first element (1), the permutations will be: [1]
3. Let's add the second element (3), the permutations will be: [3,1], [1,3]
4. Let's add the third element (5), the permutations will be: [5,3,1], [3,5,1], [3,1,5], [5,1,3], [1,5,3], [1,3,5]

Let's analyze the permutations in the 3rd and 4th step. How can we generate permutations in the 4th step from the permutations of the 3rd step?

If we look closely, we will realize that when we add a new number (5), we take each permutation of the previous step and insert the new number in every position to generate the new permutations. For example, inserting '5' in different positions of [3,1] will give us the following permutations:

1. Inserting '5' before '3': [5,3,1]
2. Inserting '5' between '3' and '1': [3,5,1]
3. Inserting '5' after '1': [3,1,5]

Here is the visual representation of this algorithm:



Code

Here is what our algorithm will look like:

Java

Python3

C++

JS JS

```

1  import java.util.*;
2
3  class Permutations {
4
5      public static List<List<Integer>> findPermutations(int[] nums) {
6          List<List<Integer>> result = new ArrayList<>();
7          Queue<List<Integer>> permutations = new LinkedList<>();
8          permutations.add(new ArrayList<>());
9          for (int currentNumber : nums) {
10             // we will take all existing permutations and add the current number to create new permutations
11             int n = permutations.size();
12             for (int i = 0; i < n; i++) {
13                 List<Integer> oldPermutation = permutations.poll();
14                 // create a new permutation by adding the current number at every position
15                 for (int j = 0; j <= oldPermutation.size(); j++) {
16                     List<Integer> newPermutation = new ArrayList<Integer>(oldPermutation);
17                     newPermutation.add(j, currentNumber);
18                     if (newPermutation.size() == nums.length)
19                         result.add(newPermutation);
20                     else
21                         permutations.add(newPermutation);
22                 }
23             }
24         }
25         return result;
26     }
27
28     public static void main(String[] args) {

```

Run

Save

Reset

Time complexity

We know that there are a total of $N!$ permutations of a set with 'N' numbers. In the algorithm above, we are iterating through all of these permutations with the help of the two 'for' loops. In each iteration, we go through all the current permutations to insert a new number in them on line 17 (line 23 for C++ solution). To insert a number into a permutation of size 'N' will take $O(N)$, which makes the overall time complexity of our algorithm $O(N * N!)$.

Space complexity

All the additional space used by our algorithm is for the result list and the queue to store the intermediate permutations. If you see closely, at any time, we don't have more than $N!$ permutations between the result list and the queue. Therefore the overall space complexity to store $N!$ permutations each containing N elements will be $O(N * N!)$.

Recursive Solution

Here is the recursive algorithm following a similar approach:

Java Python3 C++ JS

```
1 import java.util.*;
2
3 class PermutationsRecursive {
4
5     public static List<List<Integer>> generatePermutations(int[] nums) {
6         List<List<Integer>> result = new ArrayList<>();
7         generatePermutationsRecursive(nums, 0, new ArrayList<Integer>(), result);
8         return result;
9     }
10
11     private static void generatePermutationsRecursive(int[] nums, int index, List<Integer> currentPermutation, List<List<Integer>> result) {
12         List<List<Integer>> result) {
13             if (index == nums.length) {
14                 result.add(currentPermutation);
15             } else {
16                 // create a new permutation by adding the current number at every position
17                 for (int i = 0; i <= currentPermutation.size(); i++) {
18                     List<Integer> newPermutation = new ArrayList<Integer>(currentPermutation);
19                     newPermutation.add(i, nums[index]);
20                     generatePermutationsRecursive(nums, index + 1, newPermutation, result);
21                 }
22             }
23         }
24
25         public static void main(String[] args) {
26             List<List<Integer>> result = PermutationsRecursive.generatePermutations(new int[] { 1, 3, 5 });
27             System.out.print("Here are all the permutations: " + result);
28         }
29     }
30 }
```

Run Save Reset

[← Back](#)

[Next →](#)

Subsets With Duplicates (easy)

String Permutations by changing case...

☒ Mark as Completed

[Report an Issue](#) [Ask a Question](#)