

## Grokking the Coding Interview: Patterns for Coding Questions

41% completed

### Pattern: Tree Breadth

#### First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

### Pattern: Tree Depth

#### First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

### Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

### Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

## 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, 2, 3}
- {1, 3, 2}
- {2, 1, 3}
- {2, 3, 1}
- {3, 1, 2}
- {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:

JavaPython3JS C++

```
1 const find_permutations = function(nums) {
2   result = [];
3   // TODO: Write your code here
4   return result;
5 };
6
7
8 console.log('Here are all the permutations: ${find_permutations([1, 3, 5])}')
9
```

RUNSAVERESET

### 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:

- If the given set is empty then we have only an empty permutation set: []
- Let's add the first element (1), the permutations will be: [1]
- Let's add the second element (3), the permutations will be: [3,1], [1,3]
- 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:

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

- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

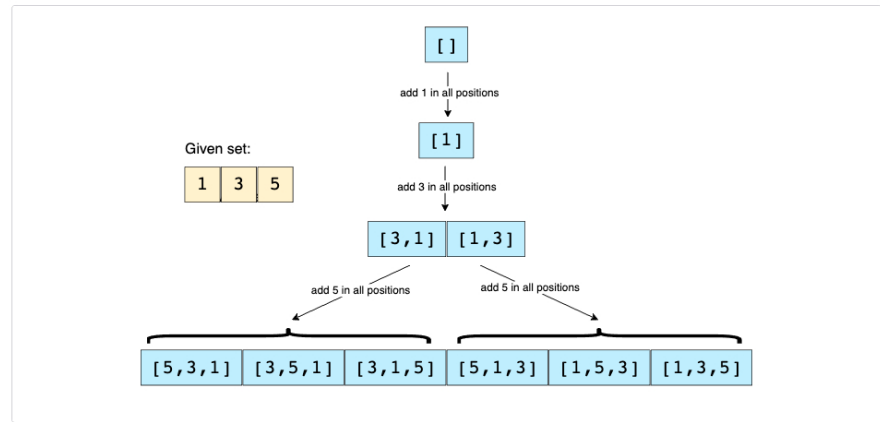
## Pattern: Top 'K' Elements

- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)
- Maximum Distinct Elements (medium)
- Sum of Elements (medium)
- Rearrange String (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: K-way merge

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted

Here is the visual representation of this algorithm:



## Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```

1  permutations.push([]);
2  for (let i = 0; i < nums.length; i++) {
3      const currentNumber = nums[i];
4      // we will take all existing permutations and add the current number to create new permutations
5      const n = permutations.length;
6      for (let p = 0; p < n; p++) {
7          const oldPermutation = permutations[p];
8          // create a new permutation by adding the current number at every position
9          for (let j = 0; j < oldPermutation.length + 1; j++) {
10             const newPermutation = oldPermutation.slice(0); // clone the permutation
11             newPermutation.splice(j, 0, currentNumber); // insert currentNumber at index 'j'
12             if (newPermutation.length === nums.length) {
13                 result.push(newPermutation);
14             } else {
15                 permutations.push(newPermutation);
16             }
17         }
18     }
19 }
20 return result;
21
22 console.log('Here are all the permutations: ', find_permutations(nums: any): any[]);
23 const result = find_permutations([1, 3, 5, 6]);
24 result.forEach((permutation) => {
25     console.log(permutation);
26 });
  
```

RUN

SAVE\*

RESET

Close

Output

5.473s

Here are all the permutations:  
 [6, 5, 3, 1]  
 [5, 6, 3, 1]  
 [5, 3, 6, 1]  
 [5, 3, 1, 6]  
 [6, 3, 5, 1]  
 [3, 6, 5, 1]  
 [3, 5, 6, 1]  
 [3, 5, 1, 6]

## 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:

Lists (Medium)

Kth Smallest Number in a Sorted Matrix (Hard)

Smallest Number Range (Hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack (Dynamic Programming)

Introduction

0/1 Knapsack (medium)

Equal Subset Sum Partition (medium)

Subset Sum (medium)

Minimum Subset Sum Difference (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

Introduction

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

Mark Course as Completed

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

JavaPython3C++JS

```
1 function generate_permutations(nums) {
2   const result = [];
3   generate_permutations_recursive(nums, 0, [], result);
4   return result;
5 }
6
7
8 function generate_permutations_recursive(nums, index, currentPermutation, result) {
9   if (index === nums.length) {
10    result.push(currentPermutation);
11   } else {
12     // create a new permutation by adding the current number at every position
13     for (let i = 0; i < currentPermutation.length + 1; i++) {
14       newPermutation = currentPermutation.slice(0); // clone the permutation
15       newPermutation.splice(i, 0, nums[index]); // insert nums[index] at index 'i'
16       generate_permutations_recursive(nums, index + 1, newPermutation, result);
17     }
18   }
19 }
20
21 console.log('Here are all the permutations:');
22 const result = generate_permutations([1, 3, 5]);
23 result.forEach((permutation) => {
24   console.log(permutation);
25 });
```

RUN

SAVE

RESET

Close

3.585s

Output

Here are all the permutations:  
[5, 3, 1]  
[3, 5, 1]  
[3, 1, 5]  
[5, 1, 3]  
[1, 5, 3]  
[1, 3, 5]

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you instead of you applying to them. [See how](#)

MARK AS COMPLETED

BackNext

Subsets With Duplicates (easy)String Permutations by changing case...

Report an Issue

Ask a Question