Grokking the Coding Interview: Patterns for Coding Questions

40% completed

Search Course

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
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)

Subsets (easy)

We'll cover the following

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

Problem Statement

Given a set with distinct elements, find all of its distinct subsets.

Example 1:

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

Example 2:

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

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const find_subsets = function(nums) {
  subsets = [];
  // TODO: Write your code here
  return subsets;
};


console.log(`Here is the list of subsets: ${find_subsets([1, 3])}`)
console.log(`Here is the list of subsets: ${find_subsets([1, 5, 3])}`)
```

RUN SAVE RESET

Solution

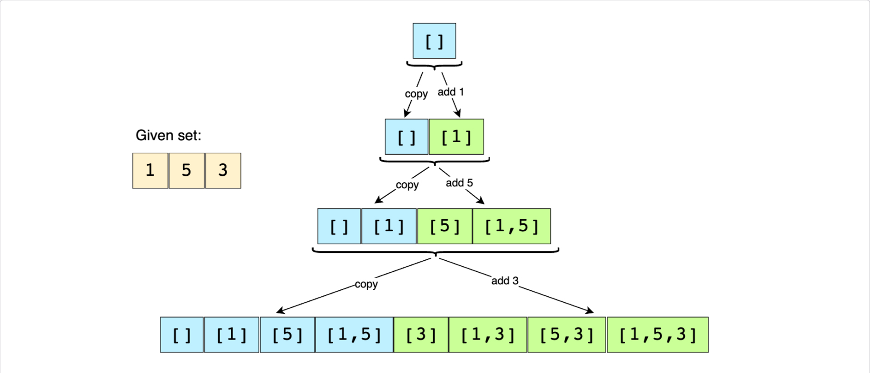
To generate all subsets of the given set, we can use the **Breadth First Search (BFS)** approach. We can start with an empty set, iterate through all numbers one-by-one, and add them to existing sets to create new subsets.

Let's take the example-2 mentioned above to go through each step of our algorithm:

Given set: [1, 5, 3]

1. Start with an empty set: [[]]
2. Add the first number (1) to all the existing subsets to create new subsets: [[], **[1]**];
3. Add the second number (5) to all the existing subsets: [[], [1], **[5], [1,5]**];
4. Add the third number (3) to all the existing subsets: [[], [1], [5], [1,5], **[3], [1,3], [5,3], [1,5,3]**].

Here is the visual representation of the above steps:



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 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

Since the input set has distinct elements, the above steps will ensure that we will not have any duplicate subsets.

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```javascript
function find_subsets(nums) {
  const subsets = [];
  // start by adding the empty subset
  subsets.push([]);
  for (i = 0; i < nums.length; i++) {
    currentNumber = nums[i];
    // we will take all existing subsets and insert the current number in them to create new subsets
    const n = subsets.length;
    for (j = 0; j < n; j++) {
      // create a new subset from the existing subset and insert the current element to it
      const set = subsets[j].slice(0); // clone the permutation
      set.push(currentNumber);
      subsets.push(set);
    }
  }

  return subsets;
}


console.log('Here is the list of subsets: ');
let result = find_subsets([1, 3]);
result.forEach((subset) => {
  console.log(subset);
});

console.log('Here is the list of subsets: ');
result = find_subsets([1, 5, 3]);
```

RUN SAVE RESET

Close

Output 2.218s

```
Here is the list of subsets:
[]
[ 1 ]
[ 3 ]
[ 1, 3 ]
Here is the list of subsets:
[]
[ 1 ]
[ 5 ]
```

Time complexity

Since, in each step, the number of subsets doubles as we add each element to all the existing subsets, the time complexity of the above algorithm is $O(2^N)$, where 'N' is the total number of elements in the input set. This also means that, in the end, we will have a total of $O(2^N)$ subsets.

Space complexity

All the additional space used by our algorithm is for the output list. Since we will have a total of $O(2^N)$ subsets, the space complexity of our algorithm is also $O(2^N)$.

☑ MARK AS COMPLETED

← Back
Introduction

Next →
Subsets With Duplicates (easy)

Report an Issue
Ask a Question