Find All subsets of an array (Power Set) LeetCode

You are tasked with developing a program that generates all possible subsets of a given set of distinct integers. A subset of a set is defined as a group of elements that can be selected from the original set, maintaining their relative order.

Approach 1: Function to find all subsets using backtracking

1. The **solve** function is the core of this approach, implementing backtracking.
2. The function takes four parameters:
    - **nums**: The input set of distinct integers.
    - **output**: The current subset being formed.
    - **index**: The current index in the input set.
    - **ans**: The vector to store the generated subsets.
3. The base case checks if the current **index** is greater than or equal to the size of the input set. If true, it means all elements have been considered, so the current **output** is added to **ans**.
4. The function first explores the scenario where the current element is excluded from the subset by calling **solve** with the same **index** but incremented by 1.
5. Then, it includes the current element in the subset, adds it to **output**, and again calls **solve** with an incremented **index**.
6. This approach systematically explores all possible combinations of including or excluding each element in the subset.
7. **Time Complexity: Exponential - O(2^n), where n is the number of elements in the input set.**
8. **Space Complexity: Linear - O(n), due to recursion stack and output vector.**

Approach 2: Function to find all subsets using backtracking (Alternative implementation)

1. The **findSubset** function is the key recursive function for this approach.
2. Similar to Approach 1, it takes four parameters: **nums**, **output**, **index**, and **ans**.
3. After adding the current **output** to **ans**, the function iterates through the remaining elements, starting from the given **index**.
4. For each element, it includes the element in the **output**, recursively calls **findSubset** with an incremented index, and then removes the last added element from the **output** (backtrack).

5. This approach is also a backtracking approach, but the logic is organized slightly differently compared to Approach 1.
6. **Time Complexity: Exponential - O(2^n), akin to the first approach.**
7. **Space Complexity: Linear - O(n), similar to the first approach.**

Approach 3: Function to find the power set using the Bitwise approach

1. This approach uses bitwise manipulation to generate subsets directly.
2. It calculates the total number of subsets as 2^n, where n is the number of elements in the input set.
3. The outer loop iterates through integers from 0 to 2^n - 1.
4. For each integer, the inner loop iterates through each bit position (element index) in the input set.
5. If the j-th bit of the current integer is set (using bitwise AND with **(1 << j)**), the corresponding element is included in the current subset.
6. The subsets are constructed and added to the final **ans**.
7. **Time Complexity: Exponential - O(n * 2^n), as each subset involves iterating over all elements.**
8. **Space Complexity: Exponential - O(n * 2^n), due to storing all generated subsets.**