

## 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++

```
1 import java.util.*;
2
3 class Subsets {
4
5     public static List<List<Integer>> findSubsets(int[] nums) {
6         List<List<Integer>> subsets = new ArrayList<>();
7         // TODO: Write your code here
8         return subsets;
9     }
10
11     public static void main(String[] args) {
12         List<List<Integer>> result = Subsets.findSubsets(new int[] { 1, 3 });
13         System.out.println("Here is the list of subsets: " + result);
14
15         result = Subsets.findSubsets(new int[] { 1, 5, 3 });
16         System.out.println("Here is the list of subsets: " + result);
17     }
18 }
19
```

Run

Save

Reset



## Solution #

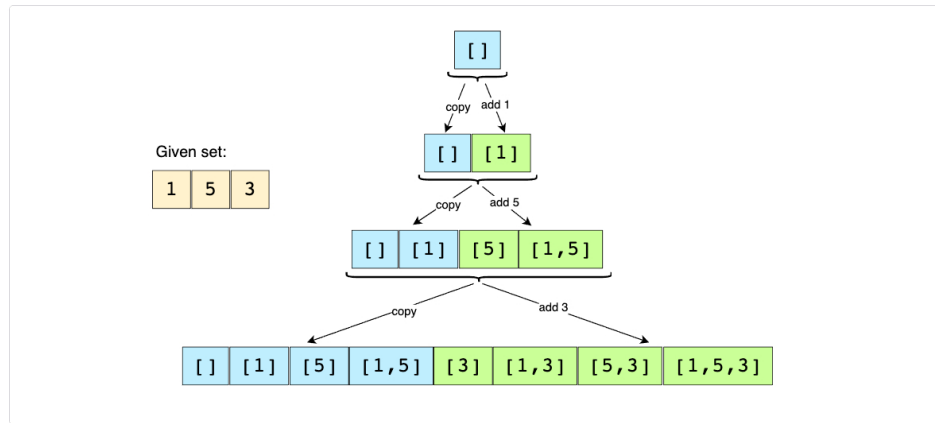
To generate all subsets of the given set, we can use the **Breadth First Search (BFS)** approach. We can start with an empty set, iterate through all numbers one-by-one, and add them to existing sets to create new subsets.

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:



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:

```

1  import java.util.*;
2
3  class Subsets {
4
5      public static List<List<Integer>> findSubsets(int[] nums) {
6          List<List<Integer>> subsets = new ArrayList<>();
7          // start by adding the empty subset
8          subsets.add(new ArrayList<>());
9          for (int currentNumber : nums) {
10             // we will take all existing subsets and insert the current number in them to create new subsets
11             int n = subsets.size();
12             for (int i = 0; i < n; i++) {
13                 // create a new subset from the existing subset and insert the current element to it
14                 List<Integer> set = new ArrayList<>(subsets.get(i));
15                 set.add(currentNumber);
16                 subsets.add(set);
17             }
18         }
19         return subsets;
20     }
21
22     public static void main(String[] args) {
23         List<List<Integer>> result = Subsets.findSubsets(new int[] { 1, 3 });
24         System.out.println("Here is the list of subsets: " + result);
25
26         result = Subsets.findSubsets(new int[] { 1, 5, 3 });
27         System.out.println("Here is the list of subsets: " + result);
28     }
29 }

```

## Time complexity #

Since, in each step, the number of subsets doubles as we add each element to all the existing subsets, therefore, we will have a total of  $O(2^N)$  subsets, where 'N' is the total number of elements in the input set. And since we construct a new subset from an existing set, therefore, the time complexity of the above algorithm will be  $O(N * 2^N)$ .

## 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, and each subset can take up to  $O(N)$  space, therefore, the space complexity of our algorithm will be  $O(N * 2^N)$ .

← Back

Next →

Introduction

Subsets With Duplicates (easy)

✓ Mark as Completed

! Report an Issue ? Ask a Question

