Subsets (easy)

We'll cover the following

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#

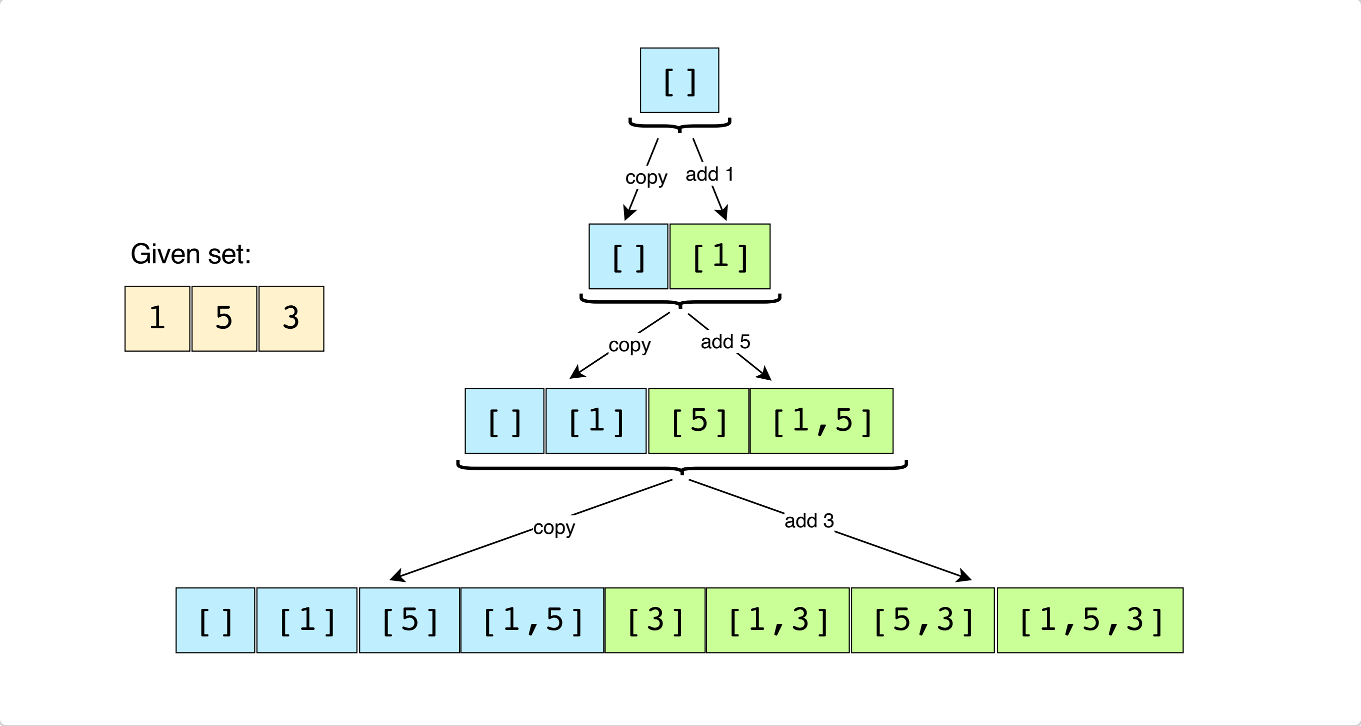
To generate all subsets of the given set, we can use the **Breadth First Search (BFS)** approach. We can start with an empty set, iterate through all numbers one-by-one, and add them to existing sets to create new subsets.

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]**].

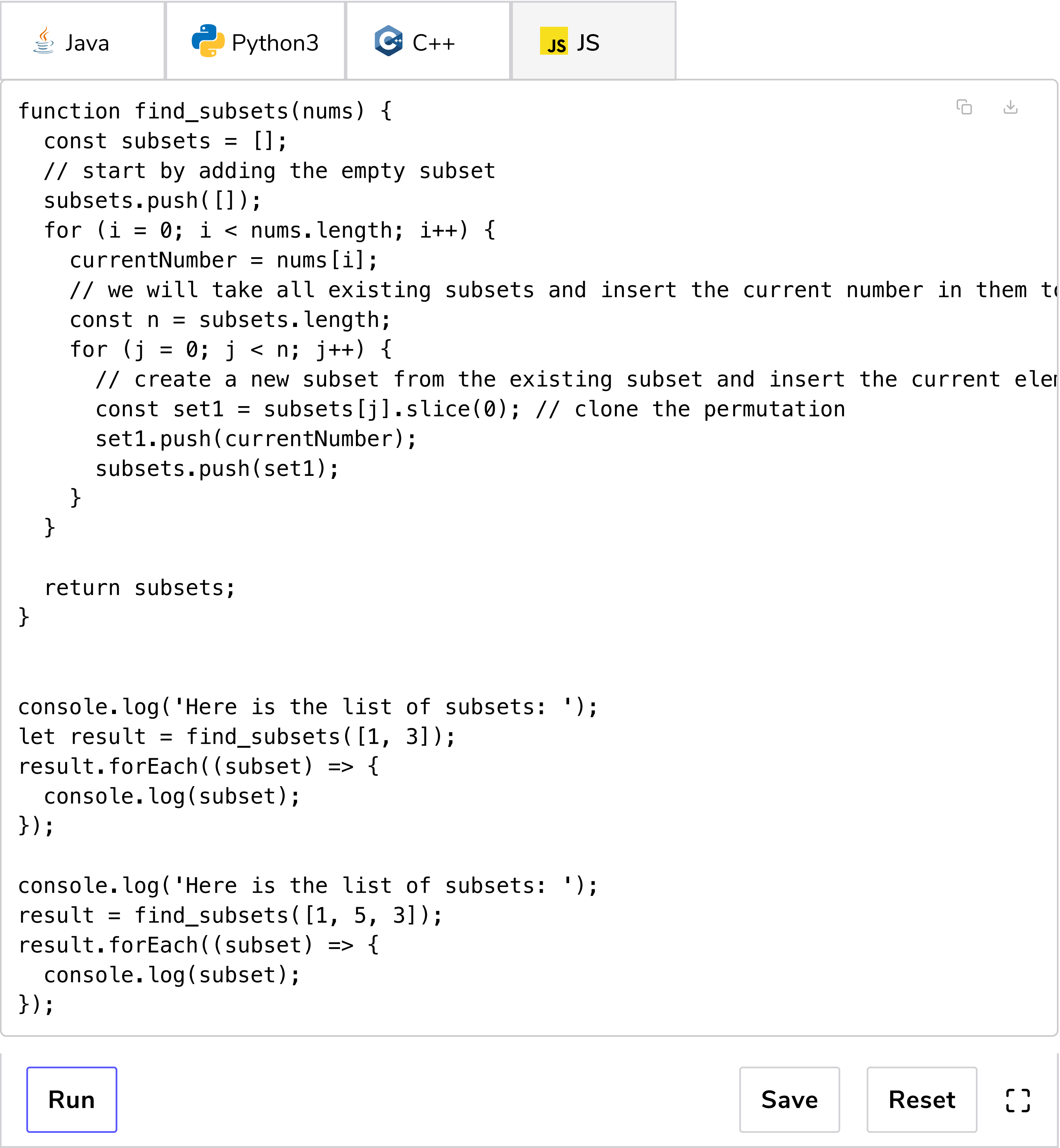
Here is the visual representation of the above steps:



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:



```
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 t
    const n = subsets.length;
    for (j = 0; j < n; j++) {
      // create a new subset from the existing subset and insert the current ele
      const set1 = subsets[j].slice(0); // clone the permutation
      set1.push(currentNumber);
      subsets.push(set1);
    }
  }

  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]);
result.forEach((subset) => {
  console.log(subset);
});
```

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

Introduction

Next →

Subsets With Duplicates (easy)

☑ Mark as Completed

Report an Issue