educative

Subsets With Duplicates (easy)

We'll cover the following

Problem Statement

Given a set of numbers that might contain duplicates, find all of its distinct subsets.

Example 1:

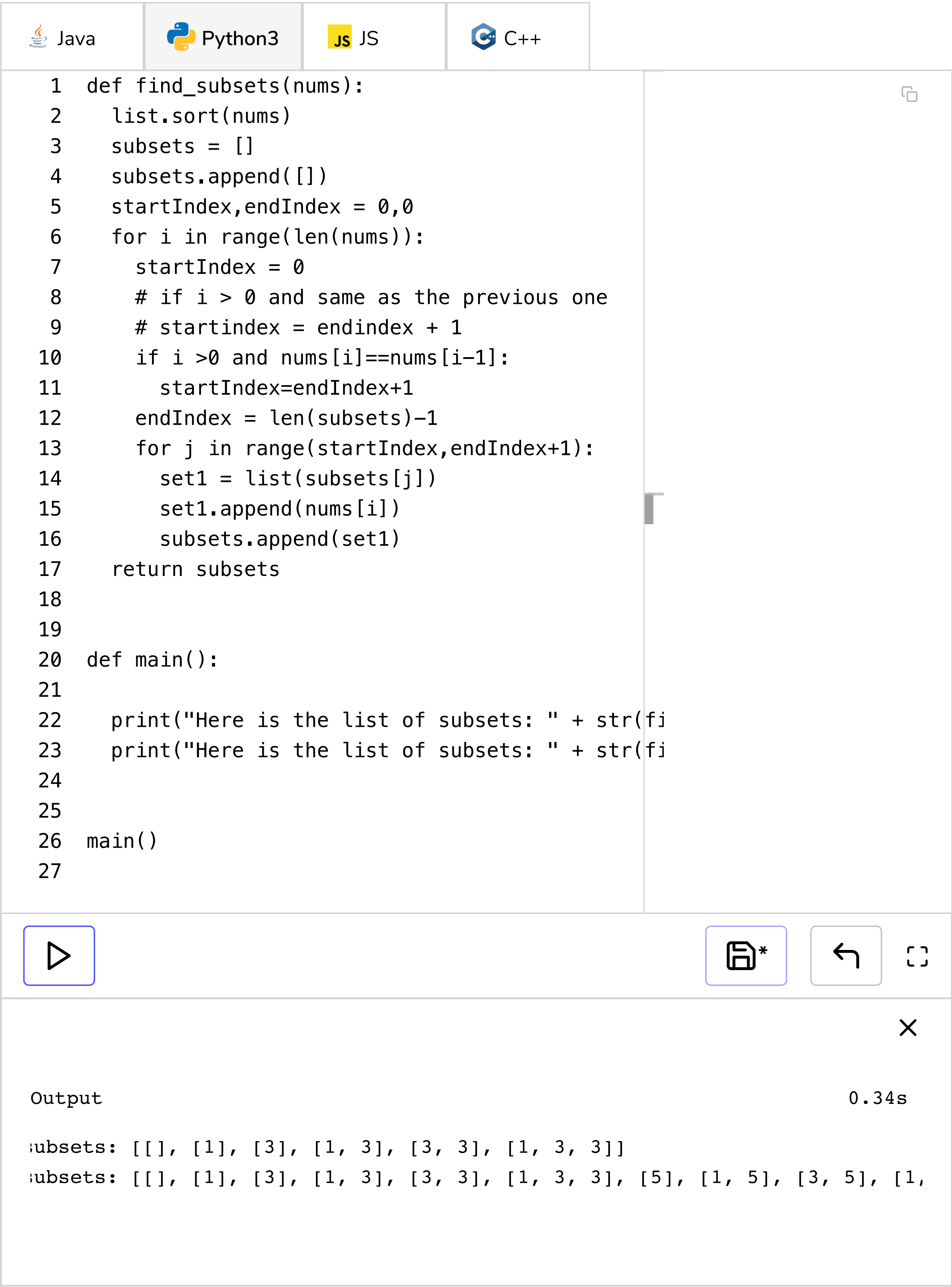
```
Input: [1, 3, 3]
Output: [], [1], [3], [1,3], [3,3], [1,3,3]
```

Example 2:

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

Try it yourself

Try solving this question here:



```python
def find_subsets(nums):
  list.sort(nums)
  subsets = []
  subsets.append([])
  startIndex,endIndex = 0,0
  for i in range(len(nums)):
    startIndex = 0
    # if i > 0 and same as the previous one
    # startindex = endindex + 1
    if i >0 and nums[i]==nums[i-1]:
      startIndex=endIndex+1
    endIndex = len(subsets)-1
    for j in range(startIndex,endIndex+1):
      set1 = list(subsets[j])
      set1.append(nums[i])
      subsets.append(set1)
  return subsets


def main():

  print("Here is the list of subsets: " + str(fi
  print("Here is the list of subsets: " + str(fi


main()
```

Output 0.34s

```
ubsets: [[], [1], [3], [1, 3], [3, 3], [1, 3, 3]]
ubsets: [[], [1], [3], [1, 3], [3, 3], [1, 3, 3], [5], [1, 5], [3, 5], [1,
```

Solution

This problem follows the Subsets (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5670249378611200) pattern and we can follow a similar **Breadth First Search (BFS)** approach. The only additional thing we need to do is handle duplicates. Since the given set can have duplicate numbers, if we follow the same approach discussed in Subsets (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5670249378611200), we will end up with duplicate subsets, which is not acceptable. To handle this, we will do two extra things:

1. Sort all numbers of the given set. This will ensure that all duplicate numbers are next to each other.
2. Follow the same BFS approach but whenever we are about to process a duplicate (i.e., when the current and the previous numbers are same), instead of adding the current number (which is a duplicate) to all the existing subsets, only add it to the subsets which were created in the previous step.

Let’s take Example-2 mentioned above to go through each step of our algorithm:

```
Given set: [1, 5, 3, 3]
Sorted set: [1, 3, 3, 5]
```

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 (3) to all the existing subsets: [[], [1], [3], [1,3]].
4. The next number (3) is a duplicate. If we add it to all existing subsets we will get:

```
    [[], [1], [3], [1,3], [3], [1,3], [3,3], [1,3,3]]
```

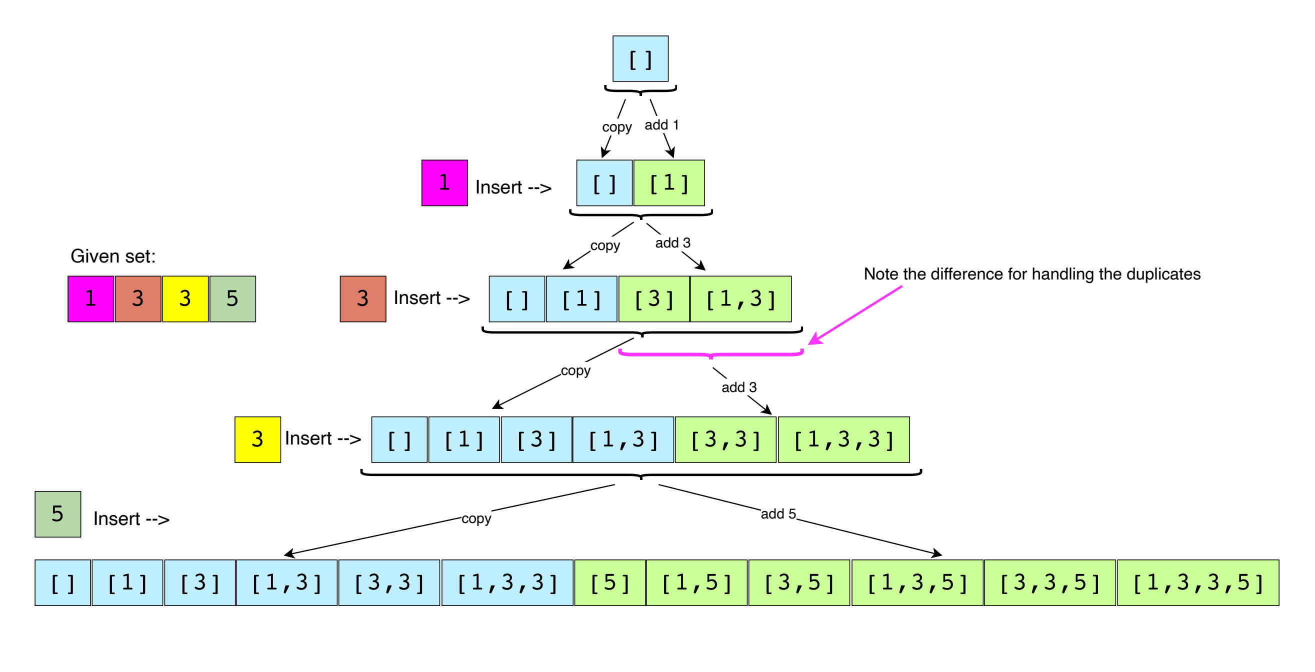
```
We got two duplicate subsets: [3], [1,3]
Whereas we only needed the new subsets: [3,3], [1,3,3]
```

To handle this instead of adding (3) to all the existing subsets, we only add it to the new subsets which were created in the previous (3rd) step:

```
    [[], [1], [3], [1,3], [3,3], [1,3,3]]
```

5. Finally, add the forth number (5) to all the existing subsets: [[], [1], [3], [1,3], [3,3], [1,3,3], [5], [1,5], [3,5], [1,3,5], [3,3,5], [1,3,3,5]]

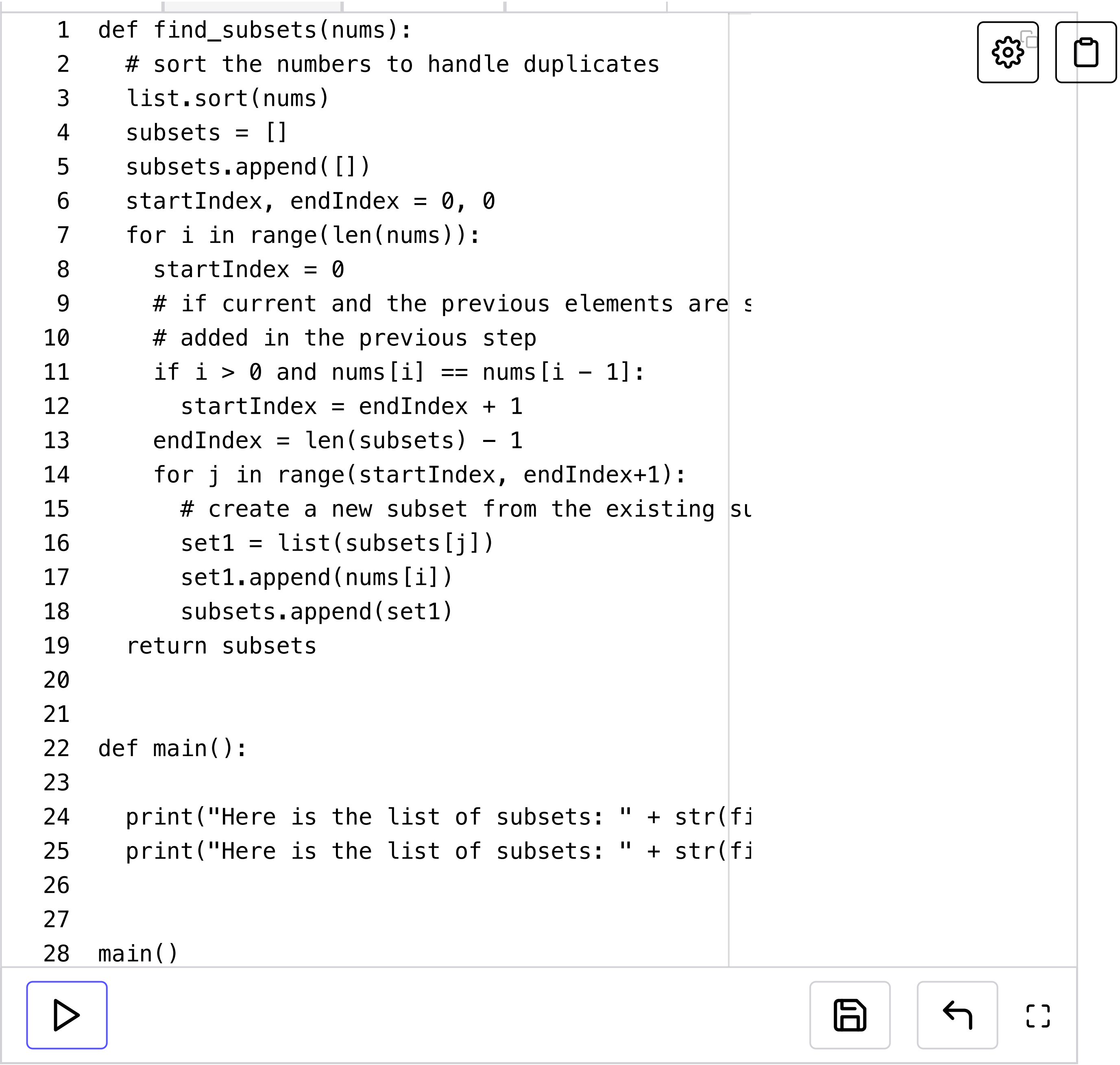
Here is the visual representation of the above steps:



Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS



```
def find_subsets(nums):
  # sort the numbers to handle duplicates
  list.sort(nums)
  subsets = []
  subsets.append([])
  startIndex, endIndex = 0, 0
  for i in range(len(nums)):
    startIndex = 0
    # if current and the previous elements are s
    # added in the previous step
    if i > 0 and nums[i] == nums[i - 1]:
      startIndex = endIndex + 1
    endIndex = len(subsets) - 1
    for j in range(startIndex, endIndex+1):
      # create a new subset from the existing su
      set1 = list(subsets[j])
      set1.append(nums[i])
      subsets.append(set1)
  return subsets


def main():

  print("Here is the list of subsets: " + str(fi
  print("Here is the list of subsets: " + str(fi


main()
```

Time complexity

Since, in each step, the number of subsets doubles (if not duplicate) 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, at most, 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

Subsets (easy)

Next →

Permutations (medium)

☑ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/subsets-with-duplicates-easy__pattern-subsets__grokking-the-coding-interview-patterns-for-coding-questions)