My algorithm starts by storing all the sets in a list of hash set of integers called sets. If the set only contains one element, it would be stored in a different list of sets called singleSets. After saving all the sets in the lists, we call the backtrack method which returns true if an answer is found and false if the answer is not found. To find the answer, it calls construct candidates that create a list of lists of sets of integers called possibleSets that hold the permutation of all the possible sets that could be created. Before the lists are actually added to possibleSets, a set of all the elements in the list is created and if the size of that set is equal to the size of the universal set, the list is added. After adding all the lists, they are sorted by size in increasing order. Therefore, the least number of sets would be first, or at position 0. If there is a set at position 0, the backtracking method returns true and prints its size and the sets in the list. Else, it combines the lists of sets with just one element and with multiple elements and runs the backtracking method again.

When I initially wrote the code, I did not make two different lists, rather just one list for all the sets. When I created all the combinations for them, it took much more time and then I realized that in most of the cases, I will not need the individual sets in most of the cases unless an answer cannot be reached without them, at which time, the lists would merge and then create the answer.

Another major change in the code I made was me using greedy algorithm. I started by using greedy algorithm where I started removing elements from the universal set that were in the biggest subset and then added the subset to the possible answer list but then I realized that would not give me the actual answer.

On the sets that start with “s-k-”, my code returns the answer for “s-k-20-30” in 4.205 seconds and the answer for “s-k-20-35” in 122.798 seconds. On the sets that start with “s-rg-”, my code returns the answer for “s-rg-40-20” in 2.822 seconds and the answer for “s-rg-63-25” in 118.427 seconds. Lastly, on the set “s-X-12-6”, it takes 0.981 seconds to return the answer.

I tried to use several approaches to prune the sets. I started by using the greedy algorithm to filter sets whose elements already existed in a temporary universal set and even if one element from the current set was not in the temporary universal set, the set would be added to the list of sets used to make permutations. This failed horribly as it would eliminate sets that had its elements mentioned before but this set would actually be a part of the answer. Then I decided to use O(N^2) method where I checked if each set was a subset of another set, if it was, I would remove it, else I would let it be. This did not work because none of the sets were subsets of each other.

# Test Cases solutions

## s-k-20-30:

```
At least 6 sets are required. The subsets required are:
[18, 3, 20, 7, 14]
[16, 1, 2, 5, 9]
[18, 4, 6, 8, 15]
[16, 17, 4, 12, 13]
[17, 3, 19, 11, 13]
[9, 10]
```

## s-rg-40-20:

```
At least 10 sets are required. The subsets required are:
[1, 2, 3]
[17, 18, 19, 4, 20]
[17, 21, 22, 7]
[1, 23, 11]
[8, 24, 25, 26, 27, 12, 15]
[28, 29]
[16, 32, 18, 13, 30]
[33, 34, 35, 36, 5, 37, 24, 9]
[32, 35, 38, 40, 10, 26]
[36, 6, 39, 29, 14, 31]
```

## s-X-12-6:

```
At least 3 sets are required. The subsets required are:
[1, 4, 7, 10]
[2, 5, 7, 8, 11]
[3, 6, 9, 12]
```