6.15 GENERATING ALL SUBSETS OF A SET

Question:

You are tasked with designing an efficient coding to generate all subsets of a given set S containing n elements. Each subset should be outputted in lexicographical order. Return a list of lists where each inner list is a subset of the given set. Additionally, find out how your coding handles duplicate elements in S.

Example: A = [1, 2, 3]
The subsets of [1, 2, 3] are: [], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]

AIM

To implement a Python program that generates all subsets (the power set) of a given set and analyze how duplicates in the input set affect the result.

ALGORITHM

1. Sort the input set to ensure lexicographical order of subsets.
2. Use backtracking to generate subsets:
   - At each step, decide whether to include the current element in the subset.
   - Recurse until all elements are considered.
3. Store each subset generated.
4. If duplicates exist in the input set, ensure they are handled by skipping over repeated elements (optional).
5. Return the complete list of subsets.

PROGRAM

```
def generate_subsets(A):
    A.sort()
    result = []
    def backtrack(start, path):
        result.append(path[:])
        for i in range(start, len(A)):
            path.append(A[i])
            backtrack(i + 1, path)
            path.pop()
    backtrack(0, [])
    return result

A = list(map(int, input("Enter set elements: ").split()))
print("Subsets:")
for subset in generate_subsets(A):
    print(subset)
```

Input:

Set: A = [1, 2, 3]

Output:

```
Enter set elements: 1 2 3
Subsets:
[]
[1]
[1, 2]
[1, 2, 3]
[1, 3]
[2]
[2, 3]
[3]
>>>
```

RESULT:

Thus, the program is successfully executed and verified to generate all subsets of a set in lexicographical order while handling duplicates appropriately.

PERFORMANCE ANALYSIS:

Time Complexity: O(2^n) where n is the number of elements, since each element can either be included or excluded.

Space Complexity: O(n) for recursion depth and temporary path storage.