**6.10 Distinct Permutations**

**Aim:** The aim is to find all distinct arrangements (permutations) of a given collection of numbers that may contain duplicates.

**Algorithm:**

1. Let the input be a list of numbers (possibly containing duplicates):  
  nums = [n₁, n₂, …, nk]

2. Sort the list nums in non-decreasing order.

3. We define a recursive backtracking procedure: permuteUnique (used, path, result)  
  - used → Boolean array of length k to mark whether an element is already included in the current arrangement  
  - path → current partial permutation  
  - result → collection of all unique permutations

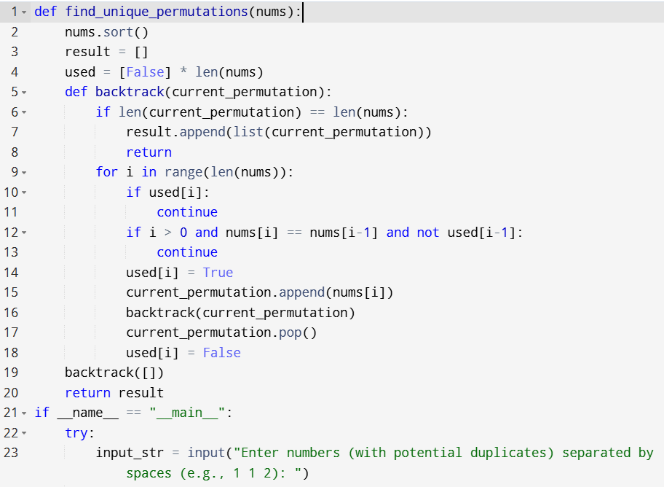
4. Base condition: - If path has length k, add a copy of path to result.

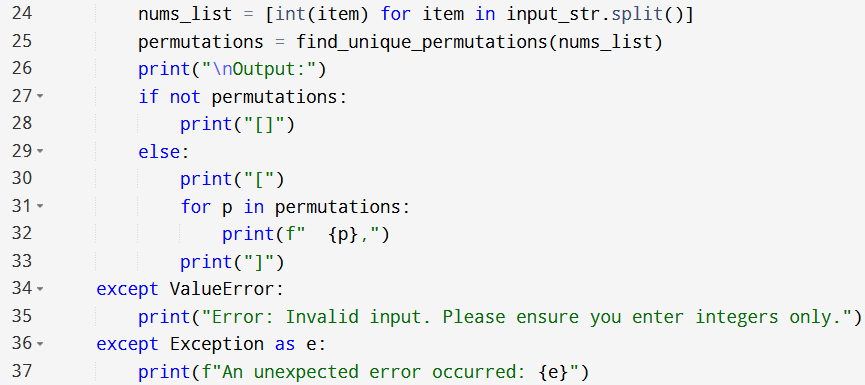
5. Recursive case:  
  - For each index i from 0 to k−1:  
    a. If used[i] = true, skip this index (already included).  
    b. If i > 0 and nums[i] = nums[i−1] and used[i−1] = false, skip this element to avoid duplicates.  
    c. Otherwise:  
      - Mark used[i] = true.  
      - Add nums[i] to path.  
      - Recursively call:  
        permuteUnique(used, path, result)  
      - Backtrack: remove nums[i] from path and set used[i] = false.

6. Initially, call the procedure as: permuteUnique([false, false, …, false], [], result)

7. The final collection result contains all distinct permutations, with duplicates removed by construction.

**Program:**

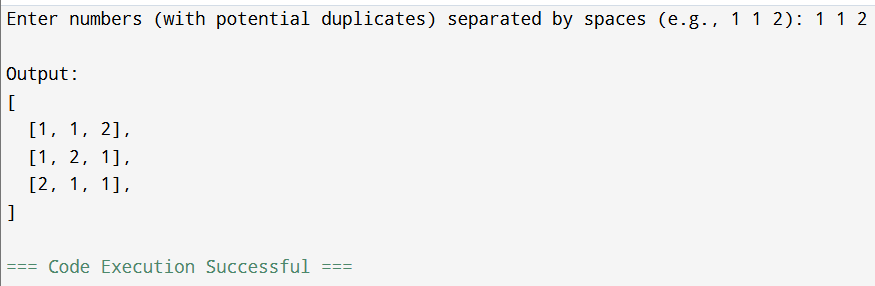
****



**Input:**

* Enter numbers (with potential duplicates) separated by spaces:

**Output:**

****

**Result:** Thus, the program is executed successfully and output is verified.

**Performance analysis:**

* Time Complexity: O(n\*n)
* Space Complexity: O(n).