3-Sum Problem Solutions (Python)

1. Hashing Based Solution

We use a set to avoid duplicate triplets. For each fixed element arr[i], we use a hashset to check if the third required element exists while iterating over arr[j].

Code:

```
class Solution:
    def threeSum(self, arr):
        n = len(arr)                                # Get array length
        st = set()                                  # Store unique triplets

        for i in range(n):                          # Fix first element
            hashset = set()                         # Store seen numbers for this i
            for j in range(i + 1, n):               # Fix second element
                third = -(arr[i] + arr[j])          # Required third number
                if third in hashset:                # If third exists, triplet foun
d
                    triplet = tuple(sorted([arr[i], arr[j], third]))
                    st.add(triplet)                 # Add to set (avoids duplicates
)
                hashset.add(arr[j])                 # Add arr[j] to hashset

        result = [list(x) for x in st]              # Convert set of tuples to list
        result.sort()                               # Sort final triplets
        return result
```

2. Two-Pointer Based Solution

We first sort the array. For each fixed arr[i], we use two pointers (j, k) to find pairs that sum to -arr[i]. This avoids using extra space and is faster.

Code:

```
class Solution:
    def triplets(self, arr):
        arr.sort()                                  # Sort array
        n = len(arr)
        matrix = []

        for i in range(n):                          # Fix first element
            if i > 0 and arr[i] == arr[i - 1]:      # Skip duplicates
                continue
            j, k = i + 1, n - 1
            while j < k:
                s = arr[i] + arr[j] + arr[k]
                if s < 0:                           # Too small, move left pointer
                    j += 1
                elif s > 0:                         # Too large, move right pointer
                    k -= 1
                else:                               # Found triplet
                    matrix.append([arr[i], arr[j], arr[k]])
                    j, k = j + 1, k - 1

                    while j < k and arr[j] == arr[j - 1]:  # Skip left duplicates
                        j += 1
                    while j < k and arr[k] == arr[k + 1]:  # Skip right duplicates
                        k -= 1
```

```
        return matrix
```

Complexity Comparison

Approach	Time Complexity	Space Complexity
Hashing	$O(n^2)$	O(n)
Two-Pointer	$O(n^2)$	O(1)