15. 3Sum

Problem Statement

1. Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

   Notice that the solution set must not contain duplicate triplets.

   Example 1:

   ```
   Input: nums = [-1,0,1,2,-1,-4]
   Output: [[-1,-1,2],[-1,0,1]]
   Explanation:
   nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
   nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
   nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
   The distinct triplets are [-1,0,1] and [-1,-1,2].
   Notice that the order of the output and the order of the triplets does not matter.
   ```

   Example 2:

   ```
   Input: nums = [0,1,1]
   Output: []
   Explanation: The only possible triplet does not sum up to 0.
   ```

   Example 3:

   ```
   Input: nums = [0,0,0]
   Output: [[0,0,0]]
   Explanation: The only possible triplet sums up to 0.
   ```

Approach

1. The function threeSum takes an input list of integers called nums and returns a list of lists, representing the triplets that satisfy the 3-sum condition.
2. The first step is to sort the input array nums in ascending order using the sort() method. Sorting the array is necessary to apply the two-pointer approach efficiently.
3. A set called triplets is initialized to store the unique triplets that satisfy the 3-sum condition. Using a set helps avoid duplicate entries in the final result.
4. The code then proceeds with a loop that iterates through each element of the array, up to the second-to-last element (len(nums) - 2). This is because we need at least three elements to form a triplet.

5. Within the loop, the current element at index i is assigned to the variable firstNum. Two pointers, j and k, are initialized. j starts from i + 1 (the element next to firstNum), and k starts from the last element of the array.
6. A while loop is used to find the pairs (secondNum and thirdNum) that can form a triplet with firstNum. The loop continues as long as j is less than k.
7. Inside the while loop, the current values at indices j and k are assigned to secondNum and thirdNum, respectively.
8. The potentialSum variable stores the sum of firstNum, secondNum, and thirdNum.
9. If potentialSum is greater than 0, it means the sum is too large. In this case, we decrement k to consider a smaller value.
10. If potentialSum is less than 0, it means the sum is too small. In this case, we increment j to consider a larger value.
11. If potentialSum is equal to 0, it means we have found a triplet that satisfies the 3-sum condition. The triplet (firstNum, secondNum, thirdNum) is added to the triplets set. Additionally, both j and k are incremented and decremented, respectively, to explore other possible combinations.
12. After the loop ends, the function returns the triplets set, which contains all the unique triplets that sum to zero.

Python code:

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        triplets = set()
        for i in range(len(nums) - 2):
            firstNum = nums[i]
            j = i + 1
            k = len(nums) - 1
            while j < k:
                secondNum  = nums[j]
                thirdNum = nums[k]

                potentialSum = firstNum + secondNum + thirdNum
                if potentialSum > 0:
                    k -= 1
                elif potentialSum < 0:
                    j += 1
                else:
                    triplets.add((firstNum , secondNum ,thirdNum))
                    j += 1
                    k -= 1
        return triplets
```