Day-12

Python DSA

Leetcode 15 3Sum

https://leetcode.com/problems/3sum/description/

Bruteforce

```
from typing import List
class Solution:
    def threeSum(self, nums:List[int])-> List[List[int]]:
        my_set=set()
        n= len(nums)
        for i in range(n):
            for j in range(i+1, n):
                for k in range(j+1,n):
                    if nums[i] + nums[j] + nums[k] ==0:
                        temp= [nums[i], nums[j] , nums[k]]
                        temp.sort()
                        my_set.add(tuple(temp))
        ans = [list(item) for item in my_set]
        return ans
nums = [-1, 0, 1, 2, -1, -4]
sol= Solution()
print(sol.threeSum(nums))
```

Time Complexity:

The time complexity is O(n3) due to the three nested loops, where n is the number of elements in the array. This makes the solution inefficient for large input sizes.

Space Complexity:

The space complexity is O(k), where k is the number of unique triplets found, stored in the set my_set.

Better

```
from typing import List
class Solution:
    def threeSum(self, nums:List[int])-> List[List[int]]:
        n= len(nums)
        result = set()
        for i in range(n):
            hashset= set()
            for j in range(i+1,n):
                third = -(nums[i] + nums[j])
                if third in hashset:
                    temp= [nums[i], nums[j], third]
                    temp.sort()
                    result.add(tuple(temp))
                hashset.add(nums[j])
        ans = list(result)
        return ans
nums = [-1, 0, 1, 2, -1, -4]
sol= Solution()
print(sol.threeSum(nums))
```

TC-O(N^2)

SC-O(N)

Optimal

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        ans = []
        n = len(nums)
        nums.sort()
        for i in range(n):
            if i != 0 and nums[i] == nums[i - 1]:
                continue

            # moving the 2 pointers
            j = i + 1
            k = n - 1
            while j < k:
                total_sum = nums[i] + nums[j] + nums[k]
                if total_sum < 0:
                    j += 1
                elif total_sum > 0:
                    k -= 1
                else:
                    temp = [nums[i], nums[j], nums[k]]
                    ans.append(temp)
                    j += 1
                    k -= 1
                    # skip the duplicates if occurred
                    while j < k and nums[j] == nums[j - 1]:
                        j += 1
                    while j < k and nums[k] == nums[k + 1]:
                        k -= 1

        return ans
```

Time Complexity:

The time complexity is O(nlogn) + O(n2). Sorting the array takes O(nlogn), and the two-pointer technique within the outer loop runs in O(n2).

Space Complexity:

The space complexity is O(1) for the two-pointer implementation (excluding the space required for the output).