Heap

1. K Closest Points to Origin (Medium)

Problem Statement:

Given an array of points where `points[i] = [xi, yi]` represents a point on the **X-Y** plane and an integer `k`, return the `k` closest points to the origin `(0, 0)`.

The distance between two points on the X-Y plane is the Euclidean distance (not squared): **[text{Distance} = sqrt{(xi - 0)^2 + (yi - 0)^2}]**

You may return the answer in **any order**. The answer is **guaranteed** to be unique (i.e., there will not be multiple points at the same distance).

Sample Input and Output:

Example 1:

```
Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance of the point (1, 3) from the origin is sqrt(10).
The distance of the point (-2, 2) from the origin is sqrt(8).
Since sqrt(8) < sqrt(10), the closest point is [-2, 2].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation:
The distance of the points from the origin are:
(3, 3) -> sqrt(18)
(5, -1) -> sqrt(26)
```

```
(-2, 4) -> sqrt(20)
The two closest points are (3, 3) and (-2, 4).
```

Notes:

- The output points can be in any order, as long as they are among the `k` closest.

```python
# Importing the required module
from typing import List
import heapq

class Solution:
    def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
        # Initialize an empty list to use as a max heap
        heap = []

        # Iterate over each point (x, y) in the list of points
        for (x, y) in points:
            # Calculate the negative of the squared Euclidean
            # distance from the origin
            # This is because the `heapq` module in Python
            # only supports a min-heap,
            # so we store the negative distance to simulate a max-heap
            dist = -(x*x + y*y)

            # If the heap contains fewer than k elements,
            # simply add the current point
            if len(heap) < k:
                heapq.heappush(heap, (dist, x, y))
            # Otherwise, if the heap already contains k elements,
            # add the current point and remove the farthest point from
            # the origin if the current one is closer
            else:
                # `heappushpop` pushes a new element and pops the
                # largest one (in terms of absolute value since distances
                # are negative here) in a single operation
                heapq.heappushpop(heap, (dist, x, y))

        # Extract the (x, y) coordinates from the heap and return them as a list
        return [(x, y) for (dist, x, y) in heap]
```

```python
# Create an instance of the Solution class
solution = Solution()

# Define a sample list of points and the value of k
points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k = 2

# Call the kClosest method and print the result
result = solution.kClosest(points, k)
print("The k closest points are:", result)
```

```
The k closest points are: [(-2, 2), (0, 1)]
```