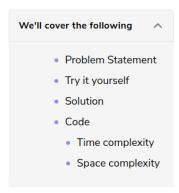
# 'K' Closest Points to the Origin (easy)



## Problem Statement #

Given an array of points in the a 2D plane, find 'K' closest points to the origin.

#### Example 1:

```
Input: points = [[1,2],[1,3]], K = 1
Output: [[1,2]]
Explanation: The Euclidean distance between (1, 2) and the origin is sqrt(5).
The Euclidean distance between (1, 3) and the origin is sqrt(10).
Since sqrt(5) < sqrt(10), therefore (1, 2) is closer to the origin.</pre>
```

### Example 2:

```
Input: point = [[1, 3], [3, 4], [2, -1]], K = 2
Output: [[1, 3], [2, -1]]
```

## Try it yourself #

Try solving this question here:

```
| Class Point:
| Class Point:
| def __init__(self, x, y):
| self,x = x
| self,y = y
| def print point(self):
| print("[" + str(self.x) + ", " + str(self.y) + "] ", end="')
| def find_closest_points(points, k):
| result = []
| # T000: Write your code here
| return result
| def main():
| result = find_closest_points([Point(1, 3), Point(3, 4), Point(2, -1)], 2)
| print("Here are the k points closest the origin: ", end="')
| for point in result:
| point.print_point()
| point.print_point()
| point.print_point()
```

## Solution |

The Euclidean distance of a point P(x,y) from the origin can be calculated through the following formula:

$$\sqrt{x^2+y^2}$$

This problem follows the Top 'K' Numbers pattern. The only difference in this problem is that we need to find the closest point (to the origin) as compared to finding the largest numbers.

Following a similar approach, we can use a **Max Heap** to find 'K' points closest to the origin. While iterating through all points, if a point (say 'P') is closer to the origin than the top point of the max-heap, we will remove that top point from the heap and add 'P' to always keep the closest points in the heap.

### Code #

Here is what our algorithm will look like:

```
👙 Java
           Python3
                        © C++
                                    Js JS
            future_
                      mport print_function
    from heapq import *
    class Point:
      def __init__(self, x, y):
        self.y = y
      def __lt__(self, other):
       return self.distance from origin() > other.distance from origin()
      def print point(self):
    def find_closest_points(points, k):
      maxHeap = []
      for i in range(k):
       heappush(maxHeap, points[i])
      for i in range(k, len(points)):
        if points[i].distance_from_origin() < maxHeap[0].distance_from_origin():</pre>
          heappop(maxHeap)
          heappush(maxHeap, points[i])
      return list(maxHeap)
    def main():
      result = find_closest_points([Point(1, 3), Point(3, 4), Point(2, -1)], 2)
      print("Here are the k points closest the origin: ", end='')
        point.print_point()
    main()
Run
                                                                                                           ::3
```

heap.

# Space complexity #

The space complexity will be O(K) because we need to store 'K' point in the heap.

