'K' Closest Points to the Origin (easy)

We'll cover the following

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.
```

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:

Java | Python3 | JS | C++

```
class Point {

  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  get_point() {
    return "[" + this.x + ", " + this.y + "] ";
  }
};

const find_closest_points = function(points, k) {
  result = [];
  // TODO: Write your code here
  return result;
};


points = find_closest_points([new Point(1, 3), new Point(3, 4), new Point(2, -1)], 2)
result = "Here are the k points closest the origin: ";
for (i=0; i < points.length; i++)
  result += points[i].get_point();
console.log(result);

```

RUN SAVE RESET

Close

Output 3.717s

```
Here are the k points closest the origin:
```

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

```
const Heap = require('./collections/heap'); //http://www.collectionsjs.com


class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  // used for max-heap
  compare(other) {
    return this.distance_from_origin() - other.distance_from_origin();
  }

  distance_from_origin() {
    // ignoring sqrt to calculate the distance
    return (this.x * this.x) + (this.y * this.y);
  }

  print_point() {
    process.stdout.write(`[${this.x}, ${this.y}] `);
  }
}

function find_closest_points(points, k) {
  const maxHeap = new Heap([], null, ((a, b) => a.compare(b)));
  // put first 'k' points in the max heap
  for (i = 0; i < k; i++) {
```

RUN | SAVE | RESET

Close

Output 5.187s

```
Here are the k points closest the origin: [1, 3] [2, -1]
```

Time complexity

The time complexity of this algorithm is $(N * logK)$ as we iterating all points and pushing them into the heap.

Space complexity

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

Back
Kth Smallest Number (easy)

Next
Connect Ropes (easy)

Mark as Completed