'K' Closest Points to the Origin (easy)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

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++

```java
import java.util.*;

class Point {
  int x;
  int y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int distFromOrigin() {
    // ignoring sqrt
    return (x * x) + (y * y);
  }
}

class KClosestPointsToOrigin {

  public static List<Point> findClosestPoints(Point[] points, int k) {
    ArrayList<Point> result = new ArrayList<>();
    // TODO: Write your code here
    return result;
  }

  public static void main(String[] args) {
    Point[] points = new Point[] { new Point(1, 3), new Point(3, 4), new Point(2, -1) };
    List<Point> result = KClosestPointsToOrigin.findClosestPoints(points, 2);
```

Run | Save | Reset

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

```java
import java.util.*;

class Point {
  int x;
  int y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int distFromOrigin() {
    // ignoring sqrt
    return (x * x) + (y * y);
  }
}

class KClosestPointsToOrigin {

  public static List<Point> findClosestPoints(Point[] points, int k) {
    PriorityQueue<Point> maxHeap = new PriorityQueue<>((p1, p2) -> p2.distFromOrigin() - p1.distFromOrigi
    // put first 'k' points in the max heap
    for (int i = 0; i < k; i++)
      maxHeap.add(points[i]);

    // go through the remaining points of the input array, if a point is closer to the origin than the to
    // of the max-heap, remove the top point from heap and add the point from the input array
    for (int i = k; i < points.length; i++) {
```

Run | Save | Reset

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

Report an Issue | Ask a Question