'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++

```python
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 = []
  # TODO: 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()


main()



```

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

```python
from __future__ import print_function
from heapq import *


class Point:

  def __init__(self, x, y):
    self.x = x
    self.y = y

  # used for max-heap
  def __lt__(self, other):
    return self.distance_from_origin() > other.distance_from_origin()

  def distance_from_origin(self):
    # ignoring sqrt to calculate the distance
    return (self.x * self.x) + (self.y * self.y)

  def print_point(self):
    print("[" + str(self.x) + ", " + str(self.y) + "] ", end='')


def find_closest_points(points, k):
  maxHeap = []
  # put first 'k' points in the max heap
  for i in range(k):
    heappush(maxHeap, points[i])

  # go through the remaining points of the input array, if a point is closer to the origin than the top p
  # of the max-heap, remove the top point from heap and add the point from the input array
  for i in range(k, len(points)):
    if points[i].distance_from_origin() < maxHeap[0].distance_from_origin():
      heappop(maxHeap)
      heappush(maxHeap, points[i])

  # the heap has 'k' points closest to the origin, return them in a list
  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='')
  for point in result:
    point.print_point()


main()
```

Run Save Reset

Time complexity

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

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)

Completed

Report an Issue Ask a Question