973. K Closest Points to Origin (/problems/k-closest-points-to-origin/)

Jan. 12, 2019 | 179.7K views

Average Rating: 2.86 (159 votes)

We have a list of `points` on the plane. Find the `K` closest points to the origin `(0, 0)`.

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

Example 1:

```
Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just [[-
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)
```

Note:

1. `1 <= K <= points.length <= 10000`

2. `-10000 < points[i][0] < 10000`
3. `-10000 < points[i][1] < 10000`

Solution

Approach 1: Sort

Intuition

Sort the points by distance, then take the closest K points.

Algorithm

There are two variants.

In Java, we find the K-th distance by creating an array of distances and then sorting them. After, we select all the points with distance less than or equal to this K-th distance.

In Python, we sort by a custom key function - namely, the distance to the origin. Afterwards, we return the first K elements of the list.

Java Python Copy

```
class Solution(object):
    def kClosest(self, points, K):
        points.sort(key = lambda P: P[0]**2 + P[1]**2)
        return points[:K]
```

Complexity Analysis

- Time Complexity: $O(N \log N)$, where N is the length of `points`.
- Space Complexity: $O(N)$.

Approach 2: Divide and Conquer

Intuition

We want an algorithm faster than $N \log N$. Clearly, the only way to do this is to use the fact that the K elements returned can be in any order -- otherwise we would be sorting which is at least $N \log N$.

Say we choose some random element `x = A[i]` and split the array into two buckets: one bucket of all the elements less than `x`, and another bucket of all the elements greater than or equal to `x`. This is known as "quickselecting by a pivot `x`".

The idea is that if we quickselect by some pivot, on average in linear time we'll reduce the problem to a problem of half the size.

Algorithm

Let's do the `work(i, j, K)` of partially sorting the subarray `(points[i], points[i+1], ...,` `points[j])` so that the smallest `K` elements of this subarray occur in the first `K` positions `(i, i+1, ..., i+K-1)`.

First, we quickselect by a random pivot element from the subarray. To do this in place, we have two pointers `i` and `j`, and move these pointers to the elements that are in the wrong bucket -- then, we swap these elements.

After, we have two buckets `[oi, i]` and `[i+1, oj]`, where `(oi, oj)` are the original `(i, j)` values when calling `work(i, j, K)`. Say the first bucket has `10` items and the second bucket has `15` items. If we were trying to partially sort say, `K = 5` items, then we only need to partially sort the first bucket: `work(oi, i, 5)`. Otherwise, if we were trying to partially sort say, `K = 17` items, then the first `10` items are already partially sorted, and we only need to partially sort the next 7 items: `work(i+1, oj, 7)`.

Java Python Articles > 973. K Closest Points to Origin Copy

```python
class Solution(object):
    def kClosest(self, points, K):
        dist = lambda i: points[i][0]**2 + points[i][1]**2

        def sort(i, j, K):
            # Partially sorts A[i:j+1] so the first K elements are
            # the smallest K elements.
            if i >= j: return

            # Put random element as A[i] - this is the pivot
            k = random.randint(i, j)
            points[i], points[k] = points[k], points[i]

            mid = partition(i, j)
            if K < mid - i + 1:
                sort(i, mid - 1, K)
            elif K > mid - i + 1:
                sort(mid + 1, j, K - (mid - i + 1))

        def partition(i, j):
            # Partition by pivot A[i], returning an index mid
            # such that A[i] <= A[mid] <= A[j] for i < mid < j.
            oi = i
            pivot = dist(i)
            i += 1

            while True:
                while i < j and dist(i) < pivot:
                    i += 1
                while i <= j and dist(j) >= pivot:
                    j -= 1
                if i >= j: break
                points[i], points[j] = points[j], points[i]

            points[oi], points[j] = points[j], points[oi]
            return j

        sort(0, len(points) - 1, K)
        return points[:K]
```

Complexity Analysis

- Time Complexity: $O(N)$ in *average case* and $O(N^2)$ in the worst case, where N is the length of `points`.
- Space Complexity: $O(N)$.

Rate this article:

Articles › 973. K Closest Points to Origin

Previous (/articles/subarray-sums-divisible-by-k/)

Next (/articles/distribute-coins-in-binary-tree/)

Comments: 95

Sort By

Type comment here... (Markdown is supported)

Preview

Post

(/kaipeng21)

KaiPeng21 (/kaipeng21) ★ 790 January 13, 2019 2:59 AM

Why not a max heap solution?

286 Share Reply

SHOW 18 REPLIES

(/harveyw)

HarveyW (/harveyw) ★ 214 January 13, 2019 12:35 AM

how about using a max heap of size K?

55 Share Reply

SHOW 12 REPLIES

(/doronin)

doronin (/doronin) ★ 67 February 10, 2019 9:53 PM

This problem can be solved with MinHeap.
O(N) to build a heap + O(Klog(N)) to extract.

60 Share Reply

SHOW 19 REPLIES

(/henrykira)

henrykira (/henrykira) ★ 198 May 16, 2019 9:47 AM

Can anyone tell me why time complexity in solution 2 is not O(N*log(K))?

16 Share Reply

SHOW 5 REPLIES

(/logicnotfound)

LogicNotFound (/logicnotfound) ★ 382 March 13, 2019 7:10 PM

Approach 1 : wont work if there are different points with same distance like (1,3) (-1,3) (1,-3)(-1,-3). So, we have 4 point with same distance. During calculation (2nd for loop) if all these points comes first in an array then we end up getting more point at kth distance than the points which are closer.

Read More

22 Share Reply

SHOW 7 REPLIES

(/ktmbdev)

ktmbdev (/ktmbdev) ★ 185 May 7, 2019 3:21 PM

WTH? I copied pasted the java solution using rand partitiioning in and it hit "Time Limit Exceeded" a few times

14 Share Reply

SHOW 1 REPLY

(/chen_o_0)

Chen_o_0 (/chen_o_0) ★ 19 January 16, 2019 4:03 AM

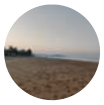
The Divide and Conquer solution is wrong.

```
while i < j:
                while i < j and dist(i) < pivot: i += 1
                while i < j and dist(j) > pivot: j -= 1
```

Read More

10 Share Reply

SHOW 3 REPLIES


(/jvargas714)

jvargas714 (/jvargas714) ★ 17 May 6, 2019 1:12 PM

why not just go with using a priority_queue (min heap).

8 Share Reply

SHOW 4 REPLIES

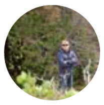
(/park29)

park29 (/park29) ★ 311 April 11, 2019 8:31 PM

Solution 2 is awesome however it is quite lengthy to implement.

7 Share Reply

SHOW 1 REPLY


(/sirik)

sirik (/sirik) ★ 46 April 11, 2019 10:54 PM

Please, fix the code in approach 2 as @jacobavelino1 (https://leetcode.com/jacobavelino1) suggested, so it doesn't give TTL.

5 Share Reply

Copyright © 2020 LeetCode Help Center (/support/) | Terms (/terms/) | Privacy (/privacy/) United States (/region/)