题目地址（822. Kth-Pair-Distance）

https://binarysearch.com/problems/Kth-Pair-Distance

入选理由

1. 能力检测二分

2. 和其他专题有联动（哪个专题呢？）

标签

- 二分

难度

- 困难

题目描述

```
Given a list of integers nums and an integer k, return the k-th (0-indexed) smallest abs(x - y) for every pair of elemen

Constraints

n ≤ 100,000 where n is the length of nums
Example 1
Input
nums = [1, 5, 3, 2]
k = 3
Output
2
Explanation
Here are all the pair distances:

abs(1 - 5) = 4
abs(1 - 3) = 2
abs(1 - 2) = 1
abs(5 - 3) = 2
abs(5 - 2) = 3
abs(3 - 2) = 1
Sorted in ascending order we have [1, 1, 2, 2, 3, 4].
```

前置知识

- 排序
- 二分法

堆（超时）

思路

堆很适合动态求极值。我在堆的专题中也说了，使用固定堆可求第 k 大的或者第 k 小的数。这道题是求第 k 小的绝对值差。于是可将所有决定值差动态加入到大顶堆中，并保持堆的大小为 k 不变。这样堆顶的就是第 k 小的绝对值差啦。

其实也可用小顶堆保存所有的绝对值差，然后弹出 k 次，最后一次弹出的就是第 k 小的绝对值差啦。

可惜的是，不管使用哪种方法都无法通过。

代码

代码支持：Python3, CPP

Python3 Code:

```
class Solution:
    def solve(self, A, k):
        A.sort()
        h = [(A[i] - A[i-1], i-1,i) for i in range(1, len(A))]
        heapq.heapify(h)

        while True:
            top, i, j = heapq.heappop(h)
            if not k: return top
            k -= 1
            if j + 1 < len(A): heapq.heappush(h, (A[j+1] - A[i], i, j + 1))
```

CPP Code:

```
int solve(vector<int>& nums, int k) {
    const int len = nums.size();
    priority_queue<int, vector<int>, greater<>> q; // 小顶堆

    for (int i = 0; i < len - 1; i++)
    {
        for (int j = i + 1; j < len; j++)
        {
```

```
            q.push(abs(nums[i] - nums[j]));
        }
    }
    if (k > q.size() - 1) return 0;
    while (k > 0)
    {
        q.pop();
        k--;
    }
    int res = q.top();
    return res;
}
```

二分法

思路

这道题是典型的计数二分。

计数二分基本就是求第 k 大（或者第 k 小）的数。其核心思想是找到一个数 x，使得小于等于 x 的数恰好有 k 个。

> 不能看出，有可能答案不止一个

对应到这道题来说就是找到一个绝对值差 diff，使得绝对值差小于等于 diff 的恰好有 k 个。

这种类型是否可用二分解决的关键在于：

如果小于等于 diff 的数恰好有 p 个：

1. p 小于 k，那么可舍弃一半解空间
2. p 大于 k，同样可舍弃一半解空间

无论如何，我们都可以舍弃一半的解空间。简单来说，就是让未知世界无机可乘。无论如何我都可以舍弃一半。

回到这道题，如果小于等于 diff 的绝对值差有大于 k 个，那么 diff 有点 大了，也就是说可以舍弃大于等于 diff 的所有值。反之也是类似，具体大家看代码吧。

最后只剩下两个问题：

- 确定解空间上下界
- 如果计算小于等于 diff 的有即可

第一个问题：下界是 0 ，上界是 max(nums) - min(min)。

第二个问题：可以使用双指针一次遍历解决。大家可以回忆趁此机会回忆一下双指针。具体地，**首先对数组排序**，然后使用右指针 j 和 左指针 i。如果 nums[j] - nums[i] 大于 diff，我们收缩 i 直到 nums[j] - nums[i] <= diff。这个时候，我们就可计算出

以索引 j 结尾的绝对值差小于等于 diff 的个数，个数就是 j - i。我们可以使用滑动窗口技巧分别计算所有的 j 的个数，并将其累加起来就是答案。

代码

代码支持：Python3,CPP,Java

Python3 Code:

```python
class Solution:
    def solve(self, A, k):
        A.sort()
        def count_not_greater(diff):
            i = ans = 0
            for j in range(1, len(A)):
                while A[j] - A[i] > diff:
                    i += 1
                ans += j - i
            return ans
        l, r = 0, A[-1] - A[0]
        k += 1 # zero based -> one based
        while l <= r:
            mid = (l + r) // 2
            if count_not_greater(mid) >= k:
                r = mid - 1
            else:
                l = mid + 1
        return l
```

CPP Code:

```cpp
bool possible(vector<int>& nums, int d, int k)
{
    const int N = nums.size();
    long count = 0; // count可能会超过2^31, 故用long存储比较稳妥
    int i = 0, j = 0;
    while (i < N || j < N)
    {
        while (j < N && nums[j] - nums[i] <= d) j++;
        count += j - i - 1;
        i++;
    }
    return count >= k;
};
int solve(vector<int>& nums, int k) {
```

```
    const int N = nums.size();
    sort(begin(nums), end(nums));
    k += 1;      //  0-indexed -> 1-indexed
    int left = 0, right = nums[N - 1] - nums[0];
    while (left < right)
    {
        int guess = left + (right - left) / 2; // guess: mid
        if (possible(nums, guess, k))
            right = guess;
        else
            left = guess + 1;
    }
    return left;
}
```

Java Code:

```
import java.util.*;

class Solution {
    public int solve(int[] nums, int k) {
        Arrays.sort(nums);
        int absMin = 0;
        int absMax = nums[nums.length-1] - nums[0];

        while (absMin <= absMax) {
            int absMid = (absMin + absMax) / 2;
            if (count_not_greater(nums, absMid) <= k) {
                absMin = absMid + 1;
            } else {
                absMax = absMid - 1;
            }
        }

        return absMin;
    }

    private long count_not_greater(int[] nums, int targetDiff) {
        long count = 0;

        int l = 0;
        for (int r=1; r<nums.length; r++) {
            while (nums[r] - nums[l] > targetDiff) {
                l++;
            }

            count += r - l;
        }
        return count;
```

```
    }
}
```

复杂度分析

令 n 为数组长度。

- 时间复杂度：由于进行了排序， 因此时间复杂度大约是 $O(nlogn)$
- 空间复杂度：取决于排序的空间消耗

思考

- 解空间的值并不都是原数组的差值（diff），那么二分能够保证答案的 diff（代码中最后返回的 l） 一定存在于原数组中么？提示：我们使用的是最左二分。

力扣的小伙伴可以**关注我**，这样就会第一时间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

上一页 下一页