Find K-th Smallest Pair Distance

Given an integer array, return the k-th smallest **distance** among all the pairs. The distance of a pair (A, B) is defined as the absolute difference between A and B.

Example 1:

```
Input:
nums = [1,3,1]
k = 1
Output: 0
Explanation:
Here are all the pairs:
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
Then the 1st smallest distance pair is (1,1), and its distance is 0.
```

Note:

1. `2 .`
2. `0 .`
3. `1 .`

Solution 1

```java
class Solution {
    // Returns number of pairs with absolute difference less than or equal to mid.
    private int countPairs(int[] a, int mid) {
        int n = a.length, res = 0;
        for (int i = 0; i < n; ++i) {
            int j = i;
            while (j < n && a[j] - a[i] <= mid) j++;
            res += j - i - 1;
        }
        return res;
    }

    public int smallestDistancePair(int a[], int k) {
        int n = a.length;
        Arrays.sort(a);

        // Minimum absolute difference
        int low = a[1] - a[0];
        for (int i = 1; i < n - 1; i++)
            low = Math.min(low, a[i + 1] - a[i]);

        // Maximum absolute difference
        int high = a[n - 1] - a[0];

        // Do binary search for k-th absolute difference
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (countPairs(a, mid) < k)
                low = mid + 1;
            else
                high = mid;
        }

        return low;
    }
}
```

Improved countPairs to use binary search too:

```java
class Solution {
    // Returns index of first index of element which is greater than key
    private int upperBound(int[] a, int low, int high, int key) {
        if (a[high] <= key) return high + 1;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (key >= a[mid]) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }
        return low;
    }

    // Returns number of pairs with absolute difference less than or equal to mid.
    private int countPairs(int[] a, int mid) {
        int n = a.length, res = 0;
        for (int i = 0; i < n; i++) {
            res += upperBound(a, i, n - 1, a[i] + mid) - i - 1;
        }
        return res;
    }

    public int smallestDistancePair(int a[], int k) {
        int n = a.length;
        Arrays.sort(a);

        // Minimum absolute difference
        int low = a[1] - a[0];
        for (int i = 1; i < n - 1; i++)
            low = Math.min(low, a[i + 1] - a[i]);

        // Maximum absolute difference
        int high = a[n - 1] - a[0];

        // Do binary search for k-th absolute difference
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (countPairs(a, mid) < k)
                low = mid + 1;
            else
                high = mid;
        }

        return low;
    }
}
```

written by shawngao original link here

Solution 2

Bucket sort, O(n^2)

```
class Solution {
public:
    int smallestDistancePair(vector<int>& nums, int k) {
        int n = nums.size(), N = 1000000;
        vector<int> cnt(N, 0);
        for (int i = 0; i < n; i++) {
            for (int j = i+1; j < n; j++)
                cnt[abs(nums[i]-nums[j])]++;
        }
        for (int i = 0; i < N; i++) {
            if (cnt[i] >= k) return i;
            k -= cnt[i];
        }
        return 0;
    }
};
```

Binary search, O(nlogn)

```
class Solution {
public:
    int smallestDistancePair(vector<int>& nums, int k) {
        sort(nums.begin(), nums.end());
        int n = nums.size(), low = 0, high = 1000000;
        while (low < high) {
            int mid = (low + high)/2, cnt = 0;
            for (int i = 0, j = 0; i < n; i++) {
                while (j < n && nums[j]-nums[i] <= mid) j++;
                cnt += j-i-1;
            }
            if (cnt < k)
                low = mid+1;
            else
                high = mid;
        }
        return low;
    }
};
```

written by zestypanda original link here

Solution 3

Find the smallest number ans such that ans is not smaller than at least k differences. For each test value of ans, counting the number of differences not more than ans is done in linear time using two pointer method.

```
class Solution {
public:
    int smallestDistancePair(vector<int>& a, int k) {
        sort(a.begin(), a.end());
        int n = a.size(), lo = 0, hi = a[n-1] - a[0], ans = -1;
        while (lo <= hi) {
            int cnt = 0, j = 0, md = (lo + hi)/2;
            for (int i = 0; i < n; ++i) {
                while (j < n && a[j] - a[i] <= md) ++j;
                cnt += j - i-1;
            }
            if (cnt >= k) {
                ans = md;
                hi = md - 1;
            }
            else lo = md + 1;
        }

        return ans;
    }
};
```

written by elastico original link here