K-th Smallest Prime Fraction

A sorted list `A` contains 1, plus some number of primes. Then, for every p < q in the list, we consider the fraction p/q.

What is the `K`-th smallest fraction considered? Return your answer as an array of ints, where `answer[0] = p` and `answer[1] = q`.

Examples:
Input: `A = [1, 2, 3, 5], K = 3`
Output: `[2, 5]`
Explanation:
The fractions to be considered in sorted order are:
1/5, 1/3, 2/5, 1/2, 3/5, 2/3.
The third fraction is 2/5.

Input: `A = [1, 7], K = 1`
Output: `[1, 7]`

Note:

- `A` will have length between `2` and `2000`.
- Each `A[i]` will be between `1` and `30000`.
- `K` will be between `1` and `A.length * (A.length + 1) / 2`.

Solution 1

This solution probably doesn't have the best runtime but it's really simple and easy to understand.

Says if the list is `[1, 7, 23, 29, 47]`, we can easily have this table of relationships

```
1/47  < 1/29    < 1/23 < 1/7
7/47  < 7/29    < 7/23
23/47 < 23/29
29/47
```

So now the problem becomes "find the kth smallest element of (n-1) sorted list"

Following is my implementation using PriorityQueue, running time is ~~O(nlogn)~~ O(max(n,k) * logn), space is O(n):

```
    public int[] kthSmallestPrimeFraction(int[] a, int k) {
        int n = a.length;
        // 0: numerator idx, 1: denominator idx
        PriorityQueue<int[]> pq = new PriorityQueue<>(new Comparator<int[]>() {
            @Override
            public int compare(int[] o1, int[] o2) {
                int s1 = a[o1[0]] * a[o2[1]];
                int s2 = a[o2[0]] * a[o1[1]];
                return s1 - s2;
            }
        });
        for (int i = 0; i < n-1; i++) {
            pq.add(new int[]{i, n-1});
        }
        for (int i = 0; i < k-1; i++) {
            int[] pop = pq.remove();
            int ni = pop[0];
            int di = pop[1];
            if (pop[1] - 1 > pop[0]) {
                pop[1]--;
                pq.add(pop);
            }
        }

        int[] peek = pq.peek();
        return new int[]{a[peek[0]], a[peek[1]]};
    }
```

written by itsleo original link here

Solution 2

```
def kthSmallestPrimeFraction(self, A, K):
    class Row(int):
        def __getitem__(self, j):
            return float(self) / A[~j], [int(self), A[~j]]
    return self.kthSmallest(map(Row, A), K)[1]

# copy&paste old solution from https://discuss.leetcode.com/topic/53126/o-n-from-paper-yes-o-rows
def kthSmallest(self, matrix, k):
    ...
```

It's O(n) where n is the size of A. Reusing my old O(n) solution for"Kth Smallest Element in a Sorted Matrix" again, similar to how I did it for"Find K-th Smallest Pair Distance". I build a virtual sorted matrix of entries `p/q, [p, q]`, get the K-th smallest entry, and extract and return its `[p, q]`. For example, for `A = [1, 2, 5, 7]` my fractions matrix looks like this:

```
  |  7    5    2    1
--+-----------------
1 | 1/7  1/5  1/2  1/1
2 | 2/7  2/5  2/2  2/1
5 | 5/7  5/5  5/2  5/1
7 | 7/7  7/5  7/2  7/1
```

written by stefanpochmann original link here

Solution 3

Basically, I first use binary search to find a estimation. Then use the estimation and predefined error to search for the precise result.
The running time is O(C*nlogn) + O(n) where C is a constant depending on the error.
I don't know the best way figure out the value of best error, but 1e-9 works for all tests cases.

```
class Solution {

    private static final double err = 1e-9;

    private int smaller(double [] A, double val) {
        int cnt = 0;
        // fix q as A[j], binary search p as A[i]
        for (int j = 1; j < A.length; ++j) {
            // find the last element that is smaller than val
            int lo = 0;
            int hi = j - 1;
            while (lo < hi) {
                int mid = (lo + hi + 1) / 2;
                if (A[mid] / A[j] < val) {
                    lo = mid;
                } else {
                    hi = mid - 1;
                }
            }
            if (lo == hi) {
                cnt += ((A[lo] / A[j] < val) ? lo + 1 : lo);
            }
        }
        return cnt;
    }

    public int[] kthSmallestPrimeFraction(int[] AA, int K) {
        // binary search the value
        double [] A = new double [AA.length];
        Set<Integer> nums = new HashSet<>();
        for (int i = 0; i < A.length; ++i) {
            A[i] = AA[i];
            nums.add(AA[i]);
        }
        double lo = 1.0 / 30000;
        double hi = 29999.0 / 30000;
        while (hi - lo > err) {
            double mid = (lo + hi) / 2;
            int cnt = smaller(A, mid); // how many fractions smaller than mid
            if (cnt > K - 1) {
                hi = mid;
            } else {
                lo = mid;
            }
        }

        for (int i = 0; i < A.length; ++i) {
```

```
            // search for each possible q (A[i])
            int p = (int)Math.round(lo * A[i]);
            if (p < A[i] && nums.contains(p) && Math.abs(p/A[i] - lo) < err) {
                return new int[] {p, AA[i]};
            }
        }
        // unreachable
        assert(false);
        return null;
    }
}
```

written by zhewang711 original link here