Problem 786: K-th Smallest Prime Fraction

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a sorted integer array

arr

containing

1

and

prime

numbers, where all the integers of

arr

are unique. You are also given an integer

k

.

For every

i

and

j

where

0 <= i < j < arr.length

, we consider the fraction

arr[i] / arr[j]

.

Return

the

k

th

smallest fraction considered

. Return your answer as an array of integers of size

2

, where

answer[0] == arr[i]

and

answer[1] == arr[j]

.

Example 1:

Input:

arr = [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, and 2/3. The third fraction is 2/5.

Example 2:

Input:

arr = [1,7], k = 1

Output:

[1,7]

Constraints:

2 <= arr.length <= 1000

1 <= arr[i] <= 3 * 10

4

arr[0] == 1

arr[i]

is a

prime

number for

i > 0

.

All the numbers of

arr

are

unique

and sorted in

strictly increasing

order.

1 <= k <= arr.length * (arr.length - 1) / 2

Follow up:

Can you solve the problem with better than

O(n

2

)

complexity?

Code Snippets

C++:

```
class Solution {
public:
vector<int> kthSmallestPrimeFraction(vector<int>& arr, int k) {

}
};
```

Java:

```
class Solution {
public int[] kthSmallestPrimeFraction(int[] arr, int k) {

}
}
```

Python3:

```
class Solution:
def kthSmallestPrimeFraction(self, arr: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def kthSmallestPrimeFraction(self, arr, k):
"""
:type arr: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} k
* @return {number[]}
*/
var kthSmallestPrimeFraction = function(arr, k) {

};
```

TypeScript:

```
function kthSmallestPrimeFraction(arr: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] KthSmallestPrimeFraction(int[] arr, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* kthSmallestPrimeFraction(int* arr, int arrSize, int k, int* returnSize)
{

}
```

Go:

```
func kthSmallestPrimeFraction(arr []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun kthSmallestPrimeFraction(arr: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func kthSmallestPrimeFraction(_ arr: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn kth_smallest_prime_fraction(arr: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer[]}
def kth_smallest_prime_fraction(arr, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer[]
*/
function kthSmallestPrimeFraction($arr, $k) {

}
}
```

Dart:

```
class Solution {
List<int> kthSmallestPrimeFraction(List<int> arr, int k) {

}
}
```

Scala:

```
object Solution {
def kthSmallestPrimeFraction(arr: Array[Int], k: Int): Array[Int] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec kth_smallest_prime_fraction(arr :: [integer], k :: integer) ::
[integer]
def kth_smallest_prime_fraction(arr, k) do

end
end
```

Erlang:

```
-spec kth_smallest_prime_fraction(Arr :: [integer()], K :: integer()) ->
[integer()].
kth_smallest_prime_fraction(Arr, K) ->
.
```

Racket:

```
(define/contract (kth-smallest-prime-fraction arr k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
vector<int> kthSmallestPrimeFraction(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] kthSmallestPrimeFraction(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K-th Smallest Prime Fraction
Difficulty: Medium
Tags: array, sort, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def kthSmallestPrimeFraction(self, arr: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def kthSmallestPrimeFraction(self, arr, k):
"""
:type arr: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} arr
* @param {number} k
* @return {number[]}
*/
var kthSmallestPrimeFraction = function(arr, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function kthSmallestPrimeFraction(arr: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] KthSmallestPrimeFraction(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: K-th Smallest Prime Fraction
* Difficulty: Medium
* Tags: array, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* kthSmallestPrimeFraction(int* arr, int arrSize, int k, int* returnSize)
{

}
```

Go Solution:

```
// Problem: K-th Smallest Prime Fraction
// Difficulty: Medium
// Tags: array, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func kthSmallestPrimeFraction(arr []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun kthSmallestPrimeFraction(arr: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func kthSmallestPrimeFraction(_ arr: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: K-th Smallest Prime Fraction
// Difficulty: Medium
// Tags: array, sort, search, queue, heap
//
```

```
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn kth_smallest_prime_fraction(arr: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer[]}
def kth_smallest_prime_fraction(arr, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer[]
*/
function kthSmallestPrimeFraction($arr, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> kthSmallestPrimeFraction(List<int> arr, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kthSmallestPrimeFraction(arr: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec kth_smallest_prime_fraction(arr :: [integer], k :: integer) ::
[integer]
def kth_smallest_prime_fraction(arr, k) do

end
end
```

Erlang Solution:

```
-spec kth_smallest_prime_fraction(Arr :: [integer()], K :: integer()) ->
[integer()].
kth_smallest_prime_fraction(Arr, K) ->
.
```

Racket Solution:

```
(define/contract (kth-smallest-prime-fraction arr k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```