Problem 2200: Find All K-Distant Indices in an Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and two integers

key

and

k

. A

k-distant index

is an index

i

of

nums

for which there exists at least one index

j

such that

|i - j| <= k

and

nums[j] == key

.

Return

a list of all k-distant indices sorted in

increasing order

.

Example 1:

Input:

nums = [3,4,9,1,3,9,5], key = 9, k = 1

Output:

[1,2,3,4,5,6]

Explanation:

Here,

nums[2] == key

and

nums[5] == key. - For index 0, |0 - 2| > k and |0 - 5| > k, so there is no j

where

|0 - j| <= k

and

nums[j] == key. Thus, 0 is not a k-distant index. - For index 1, |1 - 2| <= k and nums[2] == key, so 1 is a k-distant index. - For index 2, |2 - 2| <= k and nums[2] == key, so 2 is a k-distant index. - For index 3, |3 - 2| <= k and nums[2] == key, so 3 is a k-distant index. - For index 4, |4 - 5| <= k and nums[5] == key, so 4 is a k-distant index. - For index 5, |5 - 5| <= k and nums[5] == key, so 5 is a k-distant index. - For index 6, |6 - 5| <= k and nums[5] == key, so 6 is a k-distant index.

Thus, we return [1,2,3,4,5,6] which is sorted in increasing order.

Example 2:

Input:

nums = [2,2,2,2,2], key = 2, k = 2

Output:

[0,1,2,3,4]

Explanation:

For all indices i in nums, there exists some index j such that |i - j| <= k and nums[j] == key, so every index is a k-distant index. Hence, we return [0,1,2,3,4].

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

key

is an integer from the array

nums

.

1 <= k <= nums.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> findKDistantIndices(vector<int>& nums, int key, int k) {

}
};
```

Java:

```
class Solution {
public List<Integer> findKDistantIndices(int[] nums, int key, int k) {

}
}
```

Python3:

```
class Solution:
def findKDistantIndices(self, nums: List[int], key: int, k: int) ->
List[int]:
```

Python:

```
class Solution(object):
def findKDistantIndices(self, nums, key, k):
"""
:type nums: List[int]
:type key: int
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} key
* @param {number} k
* @return {number[]}
*/
var findKDistantIndices = function(nums, key, k) {

};
```

TypeScript:

```
function findKDistantIndices(nums: number[], key: number, k: number):
number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindKDistantIndices(int[] nums, int key, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findKDistantIndices(int* nums, int numsSize, int key, int k, int*
returnSize) {
```

```
}
```

Go:

```
func findKDistantIndices(nums []int, key int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun findKDistantIndices(nums: IntArray, key: Int, k: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func findKDistantIndices(_ nums: [Int], _ key: Int, _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_k_distant_indices(nums: Vec<i32>, key: i32, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} key
# @param {Integer} k
# @return {Integer[]}
def find_k_distant_indices(nums, key, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $key
* @param Integer $k
* @return Integer[]
*/
function findKDistantIndices($nums, $key, $k) {

}
}
```

Dart:

```
class Solution {
List<int> findKDistantIndices(List<int> nums, int key, int k) {

}
}
```

Scala:

```
object Solution {
def findKDistantIndices(nums: Array[Int], key: Int, k: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_k_distant_indices(nums :: [integer], key :: integer, k :: integer)
:: [integer]
def find_k_distant_indices(nums, key, k) do

end
end
```

Erlang:

```
-spec find_k_distant_indices(Nums :: [integer()], Key :: integer(), K ::
integer()) -> [integer()].
find_k_distant_indices(Nums, Key, K) ->
.
```

Racket:

```
(define/contract (find-k-distant-indices nums key k)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
* Tags: array, sort
*
* 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> findKDistantIndices(vector<int>& nums, int key, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
* Tags: array, sort
*
* 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 List<Integer> findKDistantIndices(int[] nums, int key, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find All K-Distant Indices in an Array
Difficulty: Easy
Tags: array, sort

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 findKDistantIndices(self, nums: List[int], key: int, k: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findKDistantIndices(self, nums, key, k):
"""
:type nums: List[int]
:type key: int
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
* Tags: array, sort
*
* 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[]} nums
* @param {number} key
* @param {number} k
* @return {number[]}
*/
var findKDistantIndices = function(nums, key, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
* Tags: array, sort
*
* 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 findKDistantIndices(nums: number[], key: number, k: number):
number[] {

};
```

C# Solution:

```
/*
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
```

```
* Tags: array, sort
*
* 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 IList<int> FindKDistantIndices(int[] nums, int key, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find All K-Distant Indices in an Array
* Difficulty: Easy
* Tags: array, sort
*
* 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* findKDistantIndices(int* nums, int numsSize, int key, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find All K-Distant Indices in an Array
// Difficulty: Easy
// Tags: array, sort
//
// 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 findKDistantIndices(nums []int, key int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findKDistantIndices(nums: IntArray, key: Int, k: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findKDistantIndices(_ nums: [Int], _ key: Int, _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find All K-Distant Indices in an Array
// Difficulty: Easy
// Tags: array, sort
//
// 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 find_k_distant_indices(nums: Vec<i32>, key: i32, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} key
# @param {Integer} k
# @return {Integer[]}
def find_k_distant_indices(nums, key, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $key
* @param Integer $k
* @return Integer[]
*/
function findKDistantIndices($nums, $key, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findKDistantIndices(List<int> nums, int key, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findKDistantIndices(nums: Array[Int], key: Int, k: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_k_distant_indices(nums :: [integer], key :: integer, k :: integer)
:: [integer]
def find_k_distant_indices(nums, key, k) do

end
end
```

Erlang Solution:

```
-spec find_k_distant_indices(Nums :: [integer()], Key :: integer(), K ::
integer()) -> [integer()].
find_k_distant_indices(Nums, Key, K) ->
.
```

Racket Solution:

```
(define/contract (find-k-distant-indices nums key k)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```