Problem 658: Find K Closest Elements

Problem Information

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

Problem Description

Given a

sorted

integer array

arr

, two integers

k

and

x

, return the

k

closest integers to

x

in the array. The result should also be sorted in ascending order.

An integer

a

is closer to

x

than an integer

b

if:

|a - x| < |b - x|

, or

|a - x| == |b - x|

and

a < b

Example 1:

Input:

arr = [1,2,3,4,5], k = 4, x = 3

Output:

[1,2,3,4]

Example 2:

Input:

arr = [1,1,2,3,4,5], k = 4, x = -1

Output:

[1,1,2,3]

Constraints:

1 <= k <= arr.length

1 <= arr.length <= 10

4

arr

is sorted in

ascending

order.

-10

4

<= arr[i], x <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> findClosestElements(int[] arr, int k, int x) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> FindClosestElements(int[] arr, int k, int x) {

}
}
```

C:

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

}
```

Go:

```
func findClosestElements(arr []int, k int, x int) []int {

}
```

Kotlin:

```
class Solution {
fun findClosestElements(arr: IntArray, k: Int, x: Int): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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



}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-closest-elements arr k x)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find K Closest Elements
* 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> findClosestElements(vector<int>& arr, int k, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Find K Closest Elements
* 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 List<Integer> findClosestElements(int[] arr, int k, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Find K Closest Elements
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 findClosestElements(self, arr: List[int], k: int, x: int) -> List[int]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find K Closest Elements
* 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
* @param {number} x
* @return {number[]}
*/
var findClosestElements = function(arr, k, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Find K Closest Elements
* 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 findClosestElements(arr: number[], k: number, x: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find K Closest Elements
* 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 IList<int> FindClosestElements(int[] arr, int k, int x) {

}
}
```

C Solution:

```
/*
* Problem: Find K Closest Elements
* 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* findClosestElements(int* arr, int arrSize, int k, int x, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find K Closest Elements
// 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 findClosestElements(arr []int, k int, x int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findClosestElements(arr: IntArray, k: Int, x: Int): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find K Closest Elements
// 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 find_closest_elements(arr: Vec<i32>, k: i32, x: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-closest-elements arr k x)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```