Problem 1471: The k Strongest Values in an Array

Problem Information

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

Problem Description

Given an array of integers

arr

and an integer

k

.

A value

arr[i]

is said to be stronger than a value

arr[j]

if

|arr[i] - m| > |arr[j] - m|

where

m

is the

centre

of the array.

If

|arr[i] - m| == |arr[j] - m|

, then

arr[i]

is said to be stronger than

arr[j]

if

arr[i] > arr[j]

.

Return

a list of the strongest

k

values in the array. return the answer

in any arbitrary order

.

The

centre

is the middle value in an ordered integer list. More formally, if the length of the list is n, the centre is the element in position

((n - 1) / 2)

in the sorted list

(0-indexed)

.

For

arr = [6, -3, 7, 2, 11]

,

n = 5

and the centre is obtained by sorting the array

arr = [-3, 2, 6, 7, 11]

and the centre is

arr[m]

where

m = ((5 - 1) / 2) = 2

. The centre is

6

.

For

arr = [-7, 22, 17, 3]

,

n = 4

and the centre is obtained by sorting the array

arr = [-7, 3, 17, 22]

and the centre is

arr[m]

where

m = ((4 - 1) / 2) = 1

. The centre is

3

.

Example 1:

Input:

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

Output:

[5,1]

Explanation:

Centre is 3, the elements of the array sorted by the strongest are [5,1,4,2,3]. The strongest 2 elements are [5, 1]. [1, 5] is also

accepted

answer. Please note that although |5 - 3| == |1 - 3| but 5 is stronger than 1 because 5 > 1.

Example 2:

Input:

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

Output:

[5,5]

Explanation:

Centre is 3, the elements of the array sorted by the strongest are [5,5,1,1,3]. The strongest 2 elements are [5, 5].

Example 3:

Input:

arr = [6,7,11,7,6,8], k = 5

Output:

[11,8,6,6,7]

Explanation:

Centre is 7, the elements of the array sorted by the strongest are [11,8,6,6,7,7]. Any permutation of [11,8,6,6,7] is

accepted

.

Constraints:

1 <= arr.length <= 10

5

-10

5

<= arr[i] <= 10

5

1 <= k <= arr.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (get-strongest arr k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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> getStrongest(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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 int[] getStrongest(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: The k Strongest Values in an Array
Difficulty: Medium
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 getStrongest(self, arr: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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[]} arr
* @param {number} k
* @return {number[]}
*/
var getStrongest = function(arr, k) {

};
```

TypeScript Solution:

```
/**
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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 getStrongest(arr: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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 int[] GetStrongest(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: The k Strongest Values in an Array
* Difficulty: Medium
* 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* getStrongest(int* arr, int arrSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: The k Strongest Values in an Array
// Difficulty: Medium
// 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 getStrongest(arr []int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The k Strongest Values in an Array
// Difficulty: Medium
// 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 get_strongest(arr: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-strongest arr k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```