Problem 3318: Find X-Sum of All K-Long Subarrays I

Problem Information

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

Problem Description

You are given an array

nums

of

n

integers and two integers

k

and

x

.

The

x-sum

of an array is calculated by the following procedure:

Count the occurrences of all elements in the array.

Keep only the occurrences of the top

x

most frequent elements. If two elements have the same number of occurrences, the element with the

bigger

value is considered more frequent.

Calculate the sum of the resulting array.

Note

that if an array has less than

x

distinct elements, its

x-sum

is the sum of the array.

Return an integer array

answer

of length

n - k + 1

where

answer[i]

is the

x-sum

of the

subarray

nums[i..i + k - 1]

.

Example 1:

Input:

nums = [1,1,2,2,3,4,2,3], k = 6, x = 2

Output:

[6,10,12]

Explanation:

For subarray

[1, 1, 2, 2, 3, 4]

, only elements 1 and 2 will be kept in the resulting array. Hence,

answer[0] = 1 + 1 + 2 + 2

.

For subarray

[1, 2, 2, 3, 4, 2]

, only elements 2 and 4 will be kept in the resulting array. Hence, answer[1] = 2 + 2 + 2 + 4

. Note that 4 is kept in the array since it is bigger than 3 and 1 which occur the same number of times.

For subarray

[2, 2, 3, 4, 2, 3]

, only elements 2 and 3 are kept in the resulting array. Hence,

answer[2] = 2 + 2 + 2 + 3 + 3

.

Example 2:

Input:

nums = [3,8,7,8,7,5], k = 2, x = 2

Output:

[11,15,15,15,12]

Explanation:

Since

k == x

,

answer[i]

is equal to the sum of the subarray

nums[i..i + k - 1]

.

Constraints:

1 <= n == nums.length <= 50

1 <= nums[i] <= 50

1 <= x <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] findXSum(int[] nums, int k, int x) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findXSum(self, nums, k, x):
"""
:type nums: List[int]
```

```
:type k: int
:type x: int
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function findXSum(nums: number[], k: number, x: number): number[] {

};
```

C#:

```
public class Solution {
public int[] FindXSum(int[] nums, int k, int x) {

}
}
```

C:

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

}
```

Go:

```
func findXSum(nums []int, k int, x int) []int {

}
```

Kotlin:

```
class Solution {
fun findXSum(nums: IntArray, k: Int, x: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> findXSum(List<int> nums, int k, int x) {

}
}
```

Scala:

```
object Solution {
def findXSum(nums: Array[Int], k: Int, x: Int): Array[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_x_sum(Nums :: [integer()], K :: integer(), X :: integer()) ->
[integer()].
find_x_sum(Nums, K, X) ->
.
```

Racket:

```
(define/contract (find-x-sum nums k x)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> findXSum(vector<int>& nums, int k, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
```

```
public int[] findXSum(int[] nums, int k, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Find X-Sum of All K-Long Subarrays I
Difficulty: Easy
Tags: array, hash, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findXSum(self, nums: List[int], k: int, x: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} k
* @param {number} x
* @return {number[]}
*/
var findXSum = function(nums, k, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findXSum(nums: number[], k: number, x: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
public class Solution {
public int[] FindXSum(int[] nums, int k, int x) {

}
}
```

C Solution:

```
/*
* Problem: Find X-Sum of All K-Long Subarrays I
* Difficulty: Easy
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

Go Solution:

```
// Problem: Find X-Sum of All K-Long Subarrays I
// Difficulty: Easy
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findXSum(nums []int, k int, x int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findXSum(nums: IntArray, k: Int, x: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find X-Sum of All K-Long Subarrays I
// Difficulty: Easy
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_x_sum(nums: Vec<i32>, k: i32, x: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> findXSum(List<int> nums, int k, int x) {

}
}
```

Scala Solution:

```
object Solution {
def findXSum(nums: Array[Int], k: Int, x: Int): Array[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_x_sum(Nums :: [integer()], K :: integer(), X :: integer()) ->
[integer()].
```

```
find_x_sum(Nums, K, X) ->
.
```

Racket Solution:

```
(define/contract (find-x-sum nums k x)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```