Problem 3066: Minimum Operations to Exceed Threshold Value II

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

, and an integer

k

.

You are allowed to perform some operations on

nums

, where in a single operation, you can:

Select the two

smallest

integers

x

and

y

from

nums

.

Remove

x

and

y

from

nums

.

Insert

(min(x, y) * 2 + max(x, y))

at any position in the array.

Note

that you can only apply the described operation if

nums

contains

at least

two elements.

Return the

minimum

number of operations needed so that all elements of the array are

greater than or equal to

k

.

Example 1:

Input:

nums = [2,11,10,1,3], k = 10

Output:

2

Explanation:

In the first operation, we remove elements 1 and 2, then add

1 * 2 + 2

to

nums

.

nums

becomes equal to

[4, 11, 10, 3]

.

In the second operation, we remove elements 3 and 4, then add

3 * 2 + 4

to

nums

.

nums

becomes equal to

[10, 11, 10]

.

At this stage, all the elements of nums are greater than or equal to 10 so we can stop.

It can be shown that 2 is the minimum number of operations needed so that all elements of the array are greater than or equal to 10.

Example 2:

Input:

nums = [1,1,2,4,9], k = 20

Output:

4

Explanation:

After one operation,

nums

becomes equal to

[2, 4, 9, 3]

.

After two operations,

nums

becomes equal to

[7, 4, 9]

.

After three operations,

nums

becomes equal to

[15, 9]

.

After four operations,

nums

becomes equal to

[33]

.

At this stage, all the elements of

nums

are greater than 20 so we can stop.

It can be shown that 4 is the minimum number of operations needed so that all elements of the array are greater than or equal to 20.

Constraints:

2 <= nums.length <= 2 * 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

9

The input is generated such that an answer always exists. That is, after performing some number of operations, all elements of the array are greater than or equal to

k

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minOperations(int* nums, int numsSize, int k) {

}
```

Go:

```
func minOperations(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-operations nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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 minOperations(vector<int>& nums, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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 minOperations(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Exceed Threshold Value II
Difficulty: Medium
Tags: array, 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 minOperations(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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[]} nums
* @param {number} k
* @return {number}
*/
var minOperations = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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 minOperations(nums: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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 MinOperations(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Exceed Threshold Value II
* Difficulty: Medium
* Tags: array, 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
*/

int minOperations(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Exceed Threshold Value II
// Difficulty: Medium
```

```
// Tags: array, 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 minOperations(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Exceed Threshold Value II
// Difficulty: Medium
// Tags: array, 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 min_operations(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-operations nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```