# Problem 2869: Minimum Operations to Collect Elements

## Problem Information

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

## Problem Description

You are given an array

nums

of positive integers and an integer

k

.

In one operation, you can remove the last element of the array and add it to your collection.

Return

the

minimum number of operations

needed to collect elements

1, 2, ..., k

.

Example 1:

Input:

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

Output:

4

Explanation:

After 4 operations, we collect elements 2, 4, 5, and 1, in this order. Our collection contains elements 1 and 2. Hence, the answer is 4.

Example 2:

Input:

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

Output:

5

Explanation:

After 5 operations, we collect elements 2, 4, 5, 1, and 3, in this order. Our collection contains elements 1 through 5. Hence, the answer is 5.

Example 3:

Input:

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

Output:

4

Explanation:

After 4 operations, we collect elements 1, 3, 5, and 2, in this order. Our collection contains elements 1 through 3. Hence, the answer is 4.

Constraints:

1 <= nums.length <= 50

1 <= nums[i] <= nums.length

1 <= k <= nums.length

The input is generated such that you can collect elements

1, 2, ..., k

.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**C:**

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

}
```

### Go:

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

}
```

### Kotlin:

```kotlin
class Solution {
fun minOperations(nums: List<Int>, k: Int): Int {

}
}
```

### Swift:

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

}
}
```

### Rust:

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

}
}
```

### Ruby:

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

end
```

### PHP:

```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: List[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 Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 minOperations(vector<int>& nums, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Operations to Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 minOperations(List<Integer> nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Operations to Collect Elements
Difficulty: Easy
Tags: array, hash

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 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 Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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
* @return {number}
*/
var minOperations = function(nums, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Operations to Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 minOperations(nums: number[], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Operations to Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 MinOperations(IList<int> nums, int k) {

}

}
```

### C Solution:

```
/*
* Problem: Minimum Operations to Collect Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

### Go Solution:

```
// Problem: Minimum Operations to Collect Elements
// Difficulty: Easy
// Tags: array, hash
//
// 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 minOperations(nums []int, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun minOperations(nums: List<Int>, k: Int): Int {

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Minimum Operations to Collect Elements
// Difficulty: Easy
// Tags: array, hash
//
// 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 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: List[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?)
)
```