Problem 2195: Append K Integers With Minimal Sum

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. Append

k

unique positive

integers that do

not

appear in

nums

to

nums

such that the resulting total sum is

minimum

.

Return

the sum of the

k

integers appended to

nums

.

Example 1:

Input:

nums = [1,4,25,10,25], k = 2

Output:

5

Explanation:

The two unique positive integers that do not appear in nums which we append are 2 and 3. The resulting sum of nums is 1 + 4 + 25 + 10 + 25 + 2 + 3 = 70, which is the minimum. The sum of the two integers appended is 2 + 3 = 5, so we return 5.

Example 2:

Input:

nums = [5,6], k = 6

Output:

25

Explanation:

The six unique positive integers that do not appear in nums which we append are 1, 2, 3, 4, 7, and 8. The resulting sum of nums is 5 + 6 + 1 + 2 + 3 + 4 + 7 + 8 = 36, which is the minimum. The sum of the six integers appended is 1 + 2 + 3 + 4 + 7 + 8 = 25, so we return 25.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

8

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
```

```
}
```

C:

```
long long minimalKSum(int* nums, int numsSize, int k) {

}
```

Go:

```
func minimalKSum(nums []int, k int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn minimal_k_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
```

```
# @return {Integer}
def minimal_k_sum(nums, k)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
* Tags: array, greedy, math, 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:
long long minimalKSum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
```

```
* Tags: array, greedy, math, 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 long minimalKSum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Append K Integers With Minimal Sum
Difficulty: Medium
Tags: array, greedy, math, 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 minimalKSum(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
* Tags: array, greedy, math, 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[]} nums
* @param {number} k
* @return {number}
*/
var minimalKSum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
* Tags: array, greedy, math, 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 minimalKSum(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
* Tags: array, greedy, math, 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 long MinimalKSum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Append K Integers With Minimal Sum
* Difficulty: Medium
* Tags: array, greedy, math, 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
*/

long long minimalKSum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Append K Integers With Minimal Sum
// Difficulty: Medium
// Tags: array, greedy, math, 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 minimalKSum(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Append K Integers With Minimal Sum
// Difficulty: Medium
// Tags: array, greedy, math, 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 minimal_k_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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