Problem 2656: Maximum Sum With Exactly K Elements

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

k

. Your task is to perform the following operation

exactly

k

times in order to maximize your score:

Select an element

m

from

nums

.

Remove the selected element

m

from the array.

Add a new element with a value of

m + 1

to the array.

Increase your score by

m

.

Return

the maximum score you can achieve after performing the operation exactly

k

times.

Example 1:

Input:

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

Output:

18

Explanation:

We need to choose exactly 3 elements from nums to maximize the sum. For the first iteration, we choose 5. Then sum is 5 and nums = [1,2,3,4,6] For the second iteration, we choose 6. Then sum is 5 + 6 and nums = [1,2,3,4,7] For the third iteration, we choose 7. Then sum is 5 + 6 + 7 = 18 and nums = [1,2,3,4,8] So, we will return 18. It can be proven, that 18 is the maximum answer that we can achieve.

Example 2:

Input:

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

Output:

11

Explanation:

We need to choose exactly 2 elements from nums to maximize the sum. For the first iteration, we choose 5. Then sum is 5 and nums = [5,5,6] For the second iteration, we choose 6. Then sum is 5 + 6 = 11 and nums = [5,5,7] So, we will return 11. It can be proven, that 11 is the maximum answer that we can achieve.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

1 <= k <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 maximizeSum(vector<int>& nums, int k) {
}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 maximizeSum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum With Exactly K Elements
Difficulty: Easy
Tags: array, greedy

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 maximizeSum(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 maximizeSum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 maximizeSum(nums: number[], k: number): number {
};
```

C# Solution:

```
/*
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 MaximizeSum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum With Exactly K Elements
* Difficulty: Easy
* Tags: array, greedy
*
* 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 maximizeSum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Sum With Exactly K Elements
// Difficulty: Easy
// Tags: array, greedy
//
// 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 maximizeSum(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum With Exactly K Elements
// Difficulty: Easy
// Tags: array, greedy
//
// 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 maximize_sum(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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