# Problem 2530: Maximal Score After Applying K Operations

## 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 have a

starting score

of

0

.

In one

operation

:

choose an index

i

such that

0 <= i < nums.length

,

increase your

score

by

nums[i]

, and

replace

nums[i]

with

ceil(nums[i] / 3)

.

Return

the maximum possible

score

you can attain after applying

exactly

k

operations

.

The ceiling function

ceil(val)

is the least integer greater than or equal to

val

.

Example 1:

Input:

nums = [10,10,10,10,10], k = 5

Output:

50

Explanation:

Apply the operation to each array element exactly once. The final score is 10 + 10 + 10 + 10 + 10 = 50.

Example 2:

Input:

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

Output:

17

Explanation:

You can do the following operations: Operation 1: Select i = 1, so nums becomes [1,

4

,3,3,3]. Your score increases by 10. Operation 2: Select i = 1, so nums becomes [1,

2

,3,3,3]. Your score increases by 4. Operation 3: Select i = 2, so nums becomes [1,2,

1

,3,3]. Your score increases by 3. The final score is 10 + 4 + 3 = 17.

Constraints:

1 <= nums.length, k <= 10

5

1 <= nums[i] <= 10

9

## Code Snippets

**C++:**

```
class Solution {
public:
```

```
long long maxKelements(vector<int>& nums, int k) {

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
```

```
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (max-kelements nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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:
long long maxKelements(vector<int>& nums, int k) {
```

```
}
};
```

### Java Solution:

```
/**
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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 long maxKelements(int[] nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximal Score After Applying K Operations
Difficulty: Medium
Tags: array, greedy, 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 maxKelements(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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 maxKelements = function(nums, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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 maxKelements(nums: number[], k: number): number {
```

```
};
```

### C# Solution:

```
/*
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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 long MaxKelements(int[] nums, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Maximal Score After Applying K Operations
* Difficulty: Medium
* Tags: array, greedy, 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
*/

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

}
```

### Go Solution:

```
// Problem: Maximal Score After Applying K Operations
// Difficulty: Medium
```

```
// Tags: array, greedy, 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 maxKelements(nums []int, k int) int64 {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Maximal Score After Applying K Operations
// Difficulty: Medium
// Tags: array, greedy, 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 max_kelements(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (max-kelements nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```