# Problem 2202: Maximize the Topmost Element After K Moves

## Problem Information

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

## Problem Description

You are given a

0-indexed

integer array

nums

representing the contents of a

pile

, where

nums[0]

is the topmost element of the pile.

In one move, you can perform

either

of the following:

If the pile is not empty,

remove

the topmost element of the pile.

If there are one or more removed elements,

add

any one of them back onto the pile. This element becomes the new topmost element.

You are also given an integer

k

, which denotes the total number of moves to be made.

Return

the

maximum value

of the topmost element of the pile possible after

exactly

k

moves

. In case it is not possible to obtain a non-empty pile after

k

moves, return

-1

.

Example 1:

Input:

nums = [5,2,2,4,0,6], k = 4

Output:

5

Explanation:

One of the ways we can end with 5 at the top of the pile after 4 moves is as follows: - Step 1: Remove the topmost element = 5. The pile becomes [2,2,4,0,6]. - Step 2: Remove the topmost element = 2. The pile becomes [2,4,0,6]. - Step 3: Remove the topmost element = 2. The pile becomes [4,0,6]. - Step 4: Add 5 back onto the pile. The pile becomes [5,4,0,6]. Note that this is not the only way to end with 5 at the top of the pile. It can be shown that 5 is the largest answer possible after 4 moves.

Example 2:

Input:

nums = [2], k = 1

Output:

-1

Explanation:

In the first move, our only option is to pop the topmost element of the pile. Since it is not possible to obtain a non-empty pile after one move, we return -1.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i], k <= 10

9

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
```

```
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (maximum-top nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximize the Topmost Element After K Moves
```

```
* Difficulty: Medium
* 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 maximumTop(vector<int>& nums, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximize the Topmost Element After K Moves
* Difficulty: Medium
* 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 maximumTop(int[] nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximize the Topmost Element After K Moves
Difficulty: Medium
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 maximumTop(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Maximize the Topmost Element After K Moves
* Difficulty: Medium
* 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 maximumTop = function(nums, k) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Maximize the Topmost Element After K Moves
* Difficulty: Medium
* 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 maximumTop(nums: number[], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Maximize the Topmost Element After K Moves
* Difficulty: Medium
* 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 MaximumTop(int[] nums, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Maximize the Topmost Element After K Moves
* Difficulty: Medium
* 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 maximumTop(int* nums, int numsSize, int k) {

}
```

### Go Solution:

```
// Problem: Maximize the Topmost Element After K Moves
// Difficulty: Medium
// 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 maximumTop(nums []int, k int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Maximize the Topmost Element After K Moves
// Difficulty: Medium
// 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 maximum_top(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (maximum-top nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```