Problem 1696: Jump Game VI

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 are initially standing at index

0

. In one move, you can jump at most

k

steps forward without going outside the boundaries of the array. That is, you can jump from index

i

to any index in the range

[i + 1, min(n - 1, i + k)]

inclusive

.

You want to reach the last index of the array (index

n - 1

). Your

score

is the

sum

of all

nums[j]

for each index

j

you visited in the array.

Return

the

maximum score

you can get

.

Example 1:

Input:

nums = [

1

,

-1

,-2,

4

,-7,

3

], k = 2

Output:

7

Explanation:

You can choose your jumps forming the subsequence [1,-1,4,3] (underlined above). The sum is 7.

Example 2:

Input:

nums = [

10

,-5,-2,

4

,0,

3

], k = 3

Output:

17

Explanation:

You can choose your jumps forming the subsequence [10,4,3] (underlined above). The sum is 17.

Example 3:

Input:

nums = [1,-5,-20,4,-1,3,-6,-3], k = 2

Output:

0

Constraints:

1 <= nums.length, k <= 10

5

-10

4

<= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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



}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Jump Game VI
Difficulty: Medium
Tags: array, dp, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxResult(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Jump Game VI
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

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

}
```

Go Solution:

```
// Problem: Jump Game VI
// Difficulty: Medium
// Tags: array, dp, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxResult(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Jump Game VI
// Difficulty: Medium
// Tags: array, dp, queue, heap
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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