

Problem 1959: Minimum Total Space Wasted With K Resizing Operations

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are currently designing a dynamic array. You are given a

0-indexed

integer array

nums

, where

nums[i]

is the number of elements that will be in the array at time

i

. In addition, you are given an integer

k

, the

maximum

number of times you can

resize

the array (to

any

size).

The size of the array at time

t

,

size

t

, must be at least

$\text{nums}[t]$

because there needs to be enough space in the array to hold all the elements. The

space wasted

at time

t

is defined as

size

t

- $\text{nums}[t]$

, and the

total

space wasted is the

sum

of the space wasted across every time

t

where

$0 \leq t < \text{nums.length}$

.

Return

the

minimum

total space wasted

if you can resize the array at most

k

times

.

Note:

The array can have

any size

at the start and does

not

count towards the number of resizing operations.

Example 1:

Input:

nums = [10,20], k = 0

Output:

10

Explanation:

size = [20,20]. We can set the initial size to be 20. The total wasted space is $(20 - 10) + (20 - 20) = 10$.

Example 2:

Input:

nums = [10,20,30], k = 1

Output:

10

Explanation:

size = [20,20,30]. We can set the initial size to be 20 and resize to 30 at time 2. The total wasted space is $(20 - 10) + (20 - 20) + (30 - 30) = 10$.

Example 3:

Input:

nums = [10,20,15,30,20], k = 2

Output:

15

Explanation:

size = [10,20,20,30,30]. We can set the initial size to 10, resize to 20 at time 1, and resize to 30 at time 3. The total wasted space is $(10 - 10) + (20 - 20) + (20 - 15) + (30 - 30) + (30 - 20) = 15$.

Constraints:

$1 \leq \text{nums.length} \leq 200$

$1 \leq \text{nums}[i] \leq 10$

6

$0 \leq k \leq \text{nums.length} - 1$

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

    }
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

    }
}
```

Swift:

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

    }
}
```

Rust:

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

    }
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

    }

}
```

Dart:

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

  }
}
```

Scala:

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

  }
}
```

Elixir:

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

  end

end
```

Erlang:

```

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

```

Racket:

```

(define/contract (min-space-wasted-k-resizing nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?)
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Minimum Total Space Wasted With K Resizing Operations
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 minSpaceWastedKResizing(vector<int>& nums, int k) {

    }
};

```

Java Solution:

```

/**
 * Problem: Minimum Total Space Wasted With K Resizing Operations
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 minSpaceWastedKResizing(int[] nums, int k) {

}

}

```

Python3 Solution:

```

"""
Problem: Minimum Total Space Wasted With K Resizing Operations
Difficulty: Medium
Tags: array, dp

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

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
* Problem: Minimum Total Space Wasted With K Resizing Operations
* Difficulty: Medium

```

```

* Tags: array, dp
*
* 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 minSpaceWastedKResizing = function(nums, k) {

};

```

TypeScript Solution:

```

/**
* Problem: Minimum Total Space Wasted With K Resizing Operations
* Difficulty: Medium
* Tags: array, dp
*
* 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 minSpaceWastedKResizing(nums: number[], k: number): number {

};

```

C# Solution:

```

/*
* Problem: Minimum Total Space Wasted With K Resizing Operations
* Difficulty: Medium
* Tags: array, dp
*
* 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 MinSpaceWastedKResizing(int[] nums, int k) {

    }
}

```

C Solution:

```

/*
* Problem: Minimum Total Space Wasted With K Resizing Operations
* Difficulty: Medium
* Tags: array, dp
*
* 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 minSpaceWastedKResizing(int* nums, int numsSize, int k) {

}

```

Go Solution:

```

// Problem: Minimum Total Space Wasted With K Resizing Operations
// Difficulty: Medium
// Tags: array, dp
//
// 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 minSpaceWastedKResizing(nums []int, k int) int {

}

```

Kotlin Solution:

```

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

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Minimum Total Space Wasted With K Resizing Operations
// Difficulty: Medium
// Tags: array, dp
//
// 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 min_space_wasted_k_resizing(nums: Vec<i32>, k: i32) -> i32 {

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

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

    }

}

```

Dart Solution:

```

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

  }

}

```

Scala Solution:

```

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

  }

}

```

Elixir Solution:

```

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

  end

end

```

Erlang Solution:

```

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

```

.

Racket Solution:

```
(define/contract (min-space-wasted-k-resizing nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?)
  )
```