

Problem 2735: Collecting Chocolates

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

of size

n

representing the cost of collecting different chocolates. The cost of collecting the chocolate at the index

i

is

nums[i]

. Each chocolate is of a different type, and initially, the chocolate at the index

i

is of

i

th

type.

In one operation, you can do the following with an incurred

cost

of

x

:

Simultaneously change the chocolate of

i

th

type to

$((i + 1) \bmod n)$

th

type for all chocolates.

Return

the minimum cost to collect chocolates of all types, given that you can perform as many operations as you would like.

Example 1:

Input:

nums = [20,1,15], x = 5

Output:

13

Explanation:

Initially, the chocolate types are [0,1,2]. We will buy the 1

st

type of chocolate at a cost of 1. Now, we will perform the operation at a cost of 5, and the types of chocolates will become [1,2,0]. We will buy the 2

nd

type of chocolate at a cost of 1. Now, we will again perform the operation at a cost of 5, and the chocolate types will become [2,0,1]. We will buy the 0

th

type of chocolate at a cost of 1. Thus, the total cost will become $(1 + 5 + 1 + 5 + 1) = 13$. We can prove that this is optimal.

Example 2:

Input:

nums = [1,2,3], x = 4

Output:

6

Explanation:

We will collect all three types of chocolates at their own price without performing any operations. Therefore, the total cost is $1 + 2 + 3 = 6$.

Constraints:

$1 \leq \text{nums.length} \leq 1000$

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

9

$1 \leq x \leq 10$

9

Code Snippets

C++:

```
class Solution {  
public:  
    long long minCost(vector<int>& nums, int x) {  
  
    }  
};
```

Java:

```
class Solution {  
public long minCost(int[] nums, int x) {  
  
}  
}
```

Python3:

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

Python:

```
class Solution(object):  
    def minCost(self, nums, x):
```

```
"""
:type nums: List[int]
:type x: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minCost(nums: number[], x: number): number {
}
```

C#:

```
public class Solution {
public long MinCost(int[] nums, int x) {

}
```

C:

```
long long minCost(int* nums, int numsSize, int x) {
}
```

Go:

```
func minCost(nums []int, x int) int64 {
}
```

Kotlin:

```
class Solution {  
    fun minCost(nums: IntArray, x: Int): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func minCost(_ nums: [Int], _ x: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn min_cost(nums: Vec<i32>, x: i32) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} nums  
# @param {Integer} x  
# @return {Integer}  
def min_cost(nums, x)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer $x  
     * @return Integer  
     */  
    function minCost($nums, $x) {
```

```
}
```

```
}
```

Dart:

```
class Solution {  
    int minCost(List<int> nums, int x) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def minCost(nums: Array[Int], x: Int): Long = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec min_cost(list :: [integer], x :: integer) :: integer  
    def min_cost(nums, x) do  
  
    end  
end
```

Erlang:

```
-spec min_cost(Nums :: [integer()], X :: integer()) -> integer().  
min_cost(Nums, X) ->  
.
```

Racket:

```
(define/contract (min-cost nums x)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Collecting Chocolates
 * Difficulty: Medium
 * Tags: array
 *
 * 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 minCost(vector<int>& nums, int x) {

    }
};
```

Java Solution:

```
/**
 * Problem: Collecting Chocolates
 * Difficulty: Medium
 * Tags: array
 *
 * 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 minCost(int[] nums, int x) {

}
```

Python3 Solution:

```
"""
Problem: Collecting Chocolates
```

Difficulty: Medium

Tags: array

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

Python Solution:

```
class Solution(object):

    def minCost(self, nums, x):
        """
        :type nums: List[int]
        :type x: int
        :rtype: int
        """
```

JavaScript Solution:

```
/**
 * Problem: Collecting Chocolates
 * Difficulty: Medium
 * Tags: array
 *
 * 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} x
 * @return {number}
 */
var minCost = function(nums, x) {
```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Collecting Chocolates  
 * Difficulty: Medium  
 * Tags: array  
 *  
 * 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 minCost(nums: number[], x: number): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Collecting Chocolates  
 * Difficulty: Medium  
 * Tags: array  
 *  
 * 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 MinCost(int[] nums, int x) {  
        return 0;  
    }  
}
```

C Solution:

```
/*  
 * Problem: Collecting Chocolates
```

```

* Difficulty: Medium
* Tags: array
*
* 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 minCost(int* nums, int numsSize, int x) {
}

```

Go Solution:

```

// Problem: Collecting Chocolates
// Difficulty: Medium
// Tags: array
//
// 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 minCost(nums []int, x int) int64 {
}

```

Kotlin Solution:

```

class Solution {
    fun minCost(nums: IntArray, x: Int): Long {
    }
}

```

Swift Solution:

```

class Solution {
    func minCost(_ nums: [Int], _ x: Int) -> Int {
    }
}

```

Rust Solution:

```
// Problem: Collecting Chocolates
// Difficulty: Medium
// Tags: array
//
// 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 min_cost(nums: Vec<i32>, x: i32) -> i64 {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} x
# @return {Integer}
def min_cost(nums, x)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $x
     * @return Integer
     */
    function minCost($nums, $x) {

    }
}
```

Dart Solution:

```
class Solution {  
    int minCost(List<int> nums, int x) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def minCost(nums: Array[Int], x: Int): Long = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec min_cost([integer], integer) :: integer  
  def min_cost(nums, x) do  
  
  end  
end
```

Erlang Solution:

```
-spec min_cost([integer()], integer()) -> integer().  
min_cost(Nums, X) ->  
.
```

Racket Solution:

```
(define/contract (min-cost nums x)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
)
```