Problem 1199: Minimum Time to Build Blocks

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a list of blocks, where

blocks[i] = t

means that the

i

-th block needs

t

units of time to be built. A block can only be built by exactly one worker.

A worker can either split into two workers (number of workers increases by one) or build a block then go home. Both decisions cost some time.

The time cost of spliting one worker into two workers is given as an integer

split

. Note that if two workers split at the same time, they split in parallel so the cost would be

split

.

Output the minimum time needed to build all blocks.

Initially, there is only

one

worker.

Example 1:

Input:

blocks = [1], split = 1

Output:

1

Explanation:

We use 1 worker to build 1 block in 1 time unit.

Example 2:

Input:

blocks = [1,2], split = 5

Output:

7

Explanation:

We split the worker into 2 workers in 5 time units then assign each of them to a block so the cost is 5 + max(1, 2) = 7.

Example 3:

Input:

blocks = [1,2,3], split = 1

Output:

4

Explanation:

Split 1 worker into 2, then assign the first worker to the last block and split the second worker into 2. Then, use the two unassigned workers to build the first two blocks. The cost is 1 + max(3, 1 + max(1, 2)) = 4.

Constraints:

1 <= blocks.length <= 1000

1 <= blocks[i] <= 10^5

1 <= split <= 100

Code Snippets

C++:

```
class Solution {
public:
int minBuildTime(vector<int>& blocks, int split) {

}
};
```

Java:

```
class Solution {
public int minBuildTime(int[] blocks, int split) {

}
}
```

Python3:

```
class Solution:
def minBuildTime(self, blocks: List[int], split: int) -> int:
```

Python:

```
class Solution(object):
def minBuildTime(self, blocks, split):
"""
:type blocks: List[int]
:type split: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} blocks
* @param {number} split
* @return {number}
*/
var minBuildTime = function(blocks, split) {

};
```

TypeScript:

```
function minBuildTime(blocks: number[], split: number): number {

};
```

C#:

```
public class Solution {
public int MinBuildTime(int[] blocks, int split) {

}
}
```

C:

```
int minBuildTime(int* blocks, int blocksSize, int split) {

}
```

Go:

```
func minBuildTime(blocks []int, split int) int {

}
```

Kotlin:

```
class Solution {
fun minBuildTime(blocks: IntArray, split: Int): Int {

}
}
```

Swift:

```
class Solution {
func minBuildTime(_ blocks: [Int], _ split: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_build_time(blocks: Vec<i32>, split: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} blocks
# @param {Integer} split
# @return {Integer}
def min_build_time(blocks, split)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $blocks
* @param Integer $split
* @return Integer
*/
function minBuildTime($blocks, $split) {

}
}
```

Dart:

```
class Solution {
int minBuildTime(List<int> blocks, int split) {

}
}
```

Scala:

```
object Solution {
def minBuildTime(blocks: Array[Int], split: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_build_time(blocks :: [integer], split :: integer) :: integer
def min_build_time(blocks, split) do

end
end
```

Erlang:

```
-spec min_build_time(Blocks :: [integer()], Split :: integer()) -> integer().
min_build_time(Blocks, Split) ->
.
```

Racket:

```
(define/contract (min-build-time blocks split)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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:
int minBuildTime(vector<int>& blocks, int split) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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 int minBuildTime(int[] blocks, int split) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Build Blocks
Difficulty: Hard
Tags: array, greedy, math, 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 minBuildTime(self, blocks: List[int], split: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minBuildTime(self, blocks, split):
"""
:type blocks: List[int]
:type split: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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[]} blocks
* @param {number} split
* @return {number}
*/
var minBuildTime = function(blocks, split) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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 minBuildTime(blocks: number[], split: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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 int MinBuildTime(int[] blocks, int split) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Build Blocks
* Difficulty: Hard
* Tags: array, greedy, math, 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
*/

int minBuildTime(int* blocks, int blocksSize, int split) {

}
```

Go Solution:

```
// Problem: Minimum Time to Build Blocks
// Difficulty: Hard
// Tags: array, greedy, math, 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 minBuildTime(blocks []int, split int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minBuildTime(blocks: IntArray, split: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minBuildTime(_ blocks: [Int], _ split: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Build Blocks
// Difficulty: Hard
// Tags: array, greedy, math, 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 min_build_time(blocks: Vec<i32>, split: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} blocks
# @param {Integer} split
# @return {Integer}
def min_build_time(blocks, split)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $blocks
* @param Integer $split
* @return Integer
*/
function minBuildTime($blocks, $split) {

}
}
```

Dart Solution:

```
class Solution {
int minBuildTime(List<int> blocks, int split) {

}
}
```

Scala Solution:

```
object Solution {
def minBuildTime(blocks: Array[Int], split: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_build_time(blocks :: [integer], split :: integer) :: integer
def min_build_time(blocks, split) do

end
end
```

Erlang Solution:

```
-spec min_build_time(Blocks :: [integer()], Split :: integer()) -> integer().
min_build_time(Blocks, Split) ->
.
```

Racket Solution:

```
(define/contract (min-build-time blocks split)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```