Problem 2594: Minimum Time to Repair Cars

Problem Information

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

Problem Description

You are given an integer array

ranks

representing the

ranks

of some mechanics.

ranks

i

is the rank of the

i

th

mechanic

.

A mechanic with a rank

r

can repair

n

cars in

r * n

2

minutes.

You are also given an integer

cars

representing the total number of cars waiting in the garage to be repaired.

Return

the

minimum

time taken to repair all the cars.

Note:

All the mechanics can repair the cars simultaneously.

Example 1:

Input:

ranks = [4,2,3,1], cars = 10

Output:

16

Explanation:

- The first mechanic will repair two cars. The time required is 4 * 2 * 2 = 16 minutes. - The second mechanic will repair two cars. The time required is 2 * 2 * 2 = 8 minutes. - The third mechanic will repair two cars. The time required is 3 * 2 * 2 = 12 minutes. - The fourth mechanic will repair four cars. The time required is 1 * 4 * 4 = 16 minutes. It can be proved that the cars cannot be repaired in less than 16 minutes.

Example 2:

Input:

ranks = [5,1,8], cars = 6

Output:

16

Explanation:

- The first mechanic will repair one car. The time required is 5 * 1 * 1 = 5 minutes. - The second mechanic will repair four cars. The time required is 1 * 4 * 4 = 16 minutes. - The third mechanic will repair one car. The time required is 8 * 1 * 1 = 8 minutes. It can be proved that the cars cannot be repaired in less than 16 minutes.

Constraints:

1 <= ranks.length <= 10

5

1 <= ranks[i] <= 100

1 <= cars <= 10

6

Code Snippets

C++:

```
class Solution {
public:
long long repairCars(vector<int>& ranks, int cars) {

}
};
```

Java:

```
class Solution {
public long repairCars(int[] ranks, int cars) {

}
}
```

Python3:

```
class Solution:
def repairCars(self, ranks: List[int], cars: int) -> int:
```

Python:

```
class Solution(object):
def repairCars(self, ranks, cars):
"""
:type ranks: List[int]
:type cars: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} ranks
* @param {number} cars
* @return {number}
*/
var repairCars = function(ranks, cars) {


};
```

TypeScript:

```
function repairCars(ranks: number[], cars: number): number {

};
```

C#:

```
public class Solution {
public long RepairCars(int[] ranks, int cars) {

}
}
```

C:

```
long long repairCars(int* ranks, int ranksSize, int cars) {

}
```

Go:

```
func repairCars(ranks []int, cars int) int64 {

}
```

Kotlin:

```
class Solution {
fun repairCars(ranks: IntArray, cars: Int): Long {

}
}
```

Swift:

```
class Solution {
func repairCars(_ ranks: [Int], _ cars: Int) -> Int {
}
}
```

Rust:

```
impl Solution {
pub fn repair_cars(ranks: Vec<i32>, cars: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} ranks
# @param {Integer} cars
# @return {Integer}
def repair_cars(ranks, cars)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $ranks
* @param Integer $cars
* @return Integer
*/
function repairCars($ranks, $cars) {

}
}
```

Dart:

```
class Solution {
int repairCars(List<int> ranks, int cars) {

}
}
```

Scala:

```
object Solution {
def repairCars(ranks: Array[Int], cars: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec repair_cars(ranks :: [integer], cars :: integer) :: integer
def repair_cars(ranks, cars) do

end
end
```

Erlang:

```
-spec repair_cars(Ranks :: [integer()], Cars :: integer()) -> integer().
repair_cars(Ranks, Cars) ->
.
```

Racket:

```
(define/contract (repair-cars ranks cars)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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 repairCars(vector<int>& ranks, int cars) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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 repairCars(int[] ranks, int cars) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Repair Cars
Difficulty: Medium
Tags: array, search

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 repairCars(self, ranks: List[int], cars: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def repairCars(self, ranks, cars):
"""
:type ranks: List[int]
:type cars: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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[]} ranks
* @param {number} cars
* @return {number}
*/
var repairCars = function(ranks, cars) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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 repairCars(ranks: number[], cars: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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 RepairCars(int[] ranks, int cars) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Repair Cars
* Difficulty: Medium
* Tags: array, search
*
* 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 repairCars(int* ranks, int ranksSize, int cars) {

}
```

Go Solution:

```
// Problem: Minimum Time to Repair Cars
// Difficulty: Medium
// Tags: array, search
//
// 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 repairCars(ranks []int, cars int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun repairCars(ranks: IntArray, cars: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func repairCars(_ ranks: [Int], _ cars: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Repair Cars
// Difficulty: Medium
// Tags: array, search
//
// 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 repair_cars(ranks: Vec<i32>, cars: i32) -> i64 {


}
}
```

Ruby Solution:

```
# @param {Integer[]} ranks
# @param {Integer} cars
# @return {Integer}
def repair_cars(ranks, cars)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $ranks
* @param Integer $cars
* @return Integer
*/
function repairCars($ranks, $cars) {

}
}
```

Dart Solution:

```
class Solution {
int repairCars(List<int> ranks, int cars) {

}
}
```

Scala Solution:

```
object Solution {
def repairCars(ranks: Array[Int], cars: Int): Long = {

}

}
```

Elixir Solution:

```
defmodule Solution do
@spec repair_cars(ranks :: [integer], cars :: integer) :: integer
def repair_cars(ranks, cars) do

end
end
```

Erlang Solution:

```
-spec repair_cars(Ranks :: [integer()], Cars :: integer()) -> integer().
repair_cars(Ranks, Cars) ->
.
```

Racket Solution:

```
(define/contract (repair-cars ranks cars)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```