Problem 1986: Minimum Number of Work Sessions to Finish the Tasks

Problem Information

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

Problem Description

There are

n

tasks assigned to you. The task times are represented as an integer array

tasks

of length

n

, where the

i

th

task takes

tasks[i]

hours to finish. A

work session

is when you work for

at most

sessionTime

consecutive hours and then take a break.

You should finish the given tasks in a way that satisfies the following conditions:

If you start a task in a work session, you must complete it in the

same

work session.

You can start a new task

immediately

after finishing the previous one.

You may complete the tasks in

any order

.

Given

tasks

and

sessionTime

, return

the

minimum

number of

work sessions

needed to finish all the tasks following the conditions above.

The tests are generated such that

sessionTime

is

greater

than or

equal

to the

maximum

element in

tasks[i]

.

Example 1:

Input:

tasks = [1,2,3], sessionTime = 3

Output:

2

Explanation:

You can finish the tasks in two work sessions. - First work session: finish the first and the second tasks in 1 + 2 = 3 hours. - Second work session: finish the third task in 3 hours.

Example 2:

Input:

tasks = [3,1,3,1,1], sessionTime = 8

Output:

2

Explanation:

You can finish the tasks in two work sessions. - First work session: finish all the tasks except the last one in 3 + 1 + 3 + 1 = 8 hours. - Second work session: finish the last task in 1 hour.

Example 3:

Input:

tasks = [1,2,3,4,5], sessionTime = 15

Output:

1

Explanation:

You can finish all the tasks in one work session.

Constraints:

n == tasks.length

1 <= n <= 14

1 <= tasks[i] <= 10

max(tasks[i]) <= sessionTime <= 15

Code Snippets

C++:

```
class Solution {
public:
int minSessions(vector<int>& tasks, int sessionTime) {

}
};
```

Java:

```
class Solution {
public int minSessions(int[] tasks, int sessionTime) {

}
}
```

Python3:

```
class Solution:
def minSessions(self, tasks: List[int], sessionTime: int) -> int:
```

Python:

```
class Solution(object):
def minSessions(self, tasks, sessionTime):
"""
:type tasks: List[int]
:type sessionTime: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} tasks
* @param {number} sessionTime
* @return {number}
*/
var minSessions = function(tasks, sessionTime) {

};
```

TypeScript:

```
function minSessions(tasks: number[], sessionTime: number): number {

};
```

C#:

```
public class Solution {
public int MinSessions(int[] tasks, int sessionTime) {

}
}
```

C:

```
int minSessions(int* tasks, int tasksSize, int sessionTime) {

}
```

Go:

```
func minSessions(tasks []int, sessionTime int) int {

}
```

Kotlin:

```
class Solution {
fun minSessions(tasks: IntArray, sessionTime: Int): Int {

}
}
```

Swift:

```
class Solution {
func minSessions(_ tasks: [Int], _ sessionTime: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_sessions(tasks: Vec<i32>, session_time: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tasks
# @param {Integer} session_time
# @return {Integer}
def min_sessions(tasks, session_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $tasks
* @param Integer $sessionTime
* @return Integer
*/
function minSessions($tasks, $sessionTime) {

}
}
```

Dart:

```
class Solution {
int minSessions(List<int> tasks, int sessionTime) {

}
}
```

Scala:

```
object Solution {
def minSessions(tasks: Array[Int], sessionTime: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_sessions(tasks :: [integer], session_time :: integer) :: integer
def min_sessions(tasks, session_time) do

end
end
```

Erlang:

```
-spec min_sessions(Tasks :: [integer()], SessionTime :: integer()) ->
integer().
min_sessions(Tasks, SessionTime) ->
.
```

Racket:

```
(define/contract (min-sessions tasks sessionTime)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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 minSessions(vector<int>& tasks, int sessionTime) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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 minSessions(int[] tasks, int sessionTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Work Sessions to Finish the Tasks
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 minSessions(self, tasks: List[int], sessionTime: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSessions(self, tasks, sessionTime):
"""
:type tasks: List[int]
:type sessionTime: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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[]} tasks
* @param {number} sessionTime
* @return {number}
*/
var minSessions = function(tasks, sessionTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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 minSessions(tasks: number[], sessionTime: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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 MinSessions(int[] tasks, int sessionTime) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Work Sessions to Finish the Tasks
* 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 minSessions(int* tasks, int tasksSize, int sessionTime) {

}
```

Go Solution:

```
// Problem: Minimum Number of Work Sessions to Finish the Tasks
// 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 minSessions(tasks []int, sessionTime int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSessions(tasks: IntArray, sessionTime: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSessions(_ tasks: [Int], _ sessionTime: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Work Sessions to Finish the Tasks
// 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_sessions(tasks: Vec<i32>, session_time: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} tasks
# @param {Integer} session_time
# @return {Integer}
def min_sessions(tasks, session_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tasks
* @param Integer $sessionTime
* @return Integer
*/
function minSessions($tasks, $sessionTime) {

}
}
```

Dart Solution:

```
class Solution {
int minSessions(List<int> tasks, int sessionTime) {

}

}
```

Scala Solution:

```scala
object Solution {
def minSessions(tasks: Array[Int], sessionTime: Int): Int = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec min_sessions(tasks :: [integer], session_time :: integer) :: integer
def min_sessions(tasks, session_time) do

end
end
```

Erlang Solution:

```erlang
-spec min_sessions(Tasks :: [integer()], SessionTime :: integer()) ->
integer().
min_sessions(Tasks, SessionTime) ->
.
```

Racket Solution:

```racket
(define/contract (min-sessions tasks sessionTime)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```