Problem 1723: Find Minimum Time to Finish All Jobs

Problem Information

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

Problem Description

You are given an integer array

jobs

, where

jobs[i]

is the amount of time it takes to complete the

i

th

job.

There are

k

workers that you can assign jobs to. Each job should be assigned to

exactly

one worker. The

working time

of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the

maximum working time

of any worker is

minimized

.

Return the

minimum

possible

maximum working time

of any assignment.

Example 1:

Input:

jobs = [3,2,3], k = 3

Output:

3

Explanation:

By assigning each person one job, the maximum time is 3.

Example 2:

Input:

jobs = [1,2,4,7,8], k = 2

Output:

11

Explanation:

Assign the jobs the following way: Worker 1: 1, 2, 8 (working time = 1 + 2 + 8 = 11) Worker 2: 4, 7 (working time = 4 + 7 = 11) The maximum working time is 11.

Constraints:

1 <= k <= jobs.length <= 12

1 <= jobs[i] <= 10

7

Code Snippets

C++:

```
class Solution {
public:
int minimumTimeRequired(vector<int>& jobs, int k) {

}
};
```

Java:

```
class Solution {
public int minimumTimeRequired(int[] jobs, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumTimeRequired(jobs: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumTimeRequired(int[] jobs, int k) {

}
}
```

C:

```
int minimumTimeRequired(int* jobs, int jobsSize, int k) {

}
```

Go:

```
func minimumTimeRequired(jobs []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumTimeRequired(jobs: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} jobs
# @param {Integer} k
# @return {Integer}
def minimum_time_required(jobs, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $jobs
* @param Integer $k
* @return Integer
*/
function minimumTimeRequired($jobs, $k) {

}
}
```

Dart:

```
class Solution {
int minimumTimeRequired(List<int> jobs, int k) {

}
}
```

Scala:

```
object Solution {
def minimumTimeRequired(jobs: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time_required(jobs :: [integer], k :: integer) :: integer
def minimum_time_required(jobs, k) do

end
end
```

Erlang:

```
-spec minimum_time_required(Jobs :: [integer()], K :: integer()) ->
integer().
```

```
minimum_time_required(Jobs, K) ->
.
```

Racket:

```
(define/contract (minimum-time-required jobs k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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 minimumTimeRequired(vector<int>& jobs, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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 minimumTimeRequired(int[] jobs, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Minimum Time to Finish All Jobs
Difficulty: Hard
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 minimumTimeRequired(self, jobs: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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[]} jobs
* @param {number} k
* @return {number}
*/
var minimumTimeRequired = function(jobs, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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 minimumTimeRequired(jobs: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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 MinimumTimeRequired(int[] jobs, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs
* Difficulty: Hard
* 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 minimumTimeRequired(int* jobs, int jobsSize, int k) {

}
```

Go Solution:

```
// Problem: Find Minimum Time to Finish All Jobs
// Difficulty: Hard
// 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 minimumTimeRequired(jobs []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTimeRequired(jobs: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Minimum Time to Finish All Jobs
// Difficulty: Hard
// 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 minimum_time_required(jobs: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} jobs
# @param {Integer} k
# @return {Integer}
def minimum_time_required(jobs, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $jobs
* @param Integer $k
* @return Integer
*/
function minimumTimeRequired($jobs, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTimeRequired(List<int> jobs, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTimeRequired(jobs: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time_required(jobs :: [integer], k :: integer) :: integer
def minimum_time_required(jobs, k) do

end
end
```

Erlang Solution:

```
-spec minimum_time_required(Jobs :: [integer()], K :: integer()) ->
integer().
minimum_time_required(Jobs, K) ->
```

```
.
```

Racket Solution:

```
(define/contract (minimum-time-required jobs k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```