Problem 2895: Minimum Processing Time

Problem Information

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

Problem Description

You have a certain number of processors, each having 4 cores. The number of tasks to be executed is four times the number of processors. Each task must be assigned to a unique core, and each core can only be used once.

You are given an array

processorTime

representing the time each processor becomes available and an array

tasks

representing how long each task takes to complete. Return the

minimum

time needed to complete all tasks.

Example 1:

Input:

processorTime = [8,10], tasks = [2,2,3,1,8,7,4,5]

Output:

16

Explanation:

Assign the tasks at indices 4, 5, 6, 7 to the first processor which becomes available at

time = 8

, and the tasks at indices 0, 1, 2, 3 to the second processor which becomes available at

time = 10

.

The time taken by the first processor to finish the execution of all tasks is

max(8 + 8, 8 + 7, 8 + 4, 8 + 5) = 16

.

The time taken by the second processor to finish the execution of all tasks is

max(10 + 2, 10 + 2, 10 + 3, 10 + 1) = 13

.

Example 2:

Input:

processorTime = [10,20], tasks = [2,3,1,2,5,8,4,3]

Output:

23

Explanation:

Assign the tasks at indices 1, 4, 5, 6 to the first processor and the others to the second processor.

The time taken by the first processor to finish the execution of all tasks is

max(10 + 3, 10 + 5, 10 + 8, 10 + 4) = 18

.

The time taken by the second processor to finish the execution of all tasks is

max(20 + 2, 20 + 1, 20 + 2, 20 + 3) = 23

.

Constraints:

1 <= n == processorTime.length <= 25000

1 <= tasks.length <= 10

5

0 <= processorTime[i] <= 10

9

1 <= tasks[i] <= 10

9

tasks.length == 4 * n

Code Snippets

C++:

```
class Solution {
public:
int minProcessingTime(vector<int>& processorTime, vector<int>& tasks) {

}
};
```

Java:

```
class Solution {
public int minProcessingTime(List<Integer> processorTime, List<Integer>
tasks) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minProcessingTime(processorTime: number[], tasks: number[]): number
{

};
```

C#:

```
public class Solution {
public int MinProcessingTime(IList<int> processorTime, IList<int> tasks) {

}
}
```

C:

```
int minProcessingTime(int* processorTime, int processorTimeSize, int* tasks,
int tasksSize) {

}
```

Go:

```
func minProcessingTime(processorTime []int, tasks []int) int {

}
```

Kotlin:

```
class Solution {
fun minProcessingTime(processorTime: List<Int>, tasks: List<Int>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} processor_time
# @param {Integer[]} tasks
# @return {Integer}
def min_processing_time(processor_time, tasks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $processorTime
* @param Integer[] $tasks
* @return Integer
*/
function minProcessingTime($processorTime, $tasks) {

}
}
```

Dart:

```
class Solution {
int minProcessingTime(List<int> processorTime, List<int> tasks) {

}
}
```

Scala:

```
object Solution {
def minProcessingTime(processorTime: List[Int], tasks: List[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_processing_time(processor_time :: [integer], tasks :: [integer]) ::
integer
def min_processing_time(processor_time, tasks) do

end
end
```

Erlang:

```
-spec min_processing_time(ProcessorTime :: [integer()], Tasks :: [integer()])
-> integer().
min_processing_time(ProcessorTime, Tasks) ->
.
```

Racket:

```
(define/contract (min-processing-time processorTime tasks)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Processing Time
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minProcessingTime(vector<int>& processorTime, vector<int>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Processing Time
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minProcessingTime(List<Integer> processorTime, List<Integer>
tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Processing Time
Difficulty: Medium
Tags: array, greedy, sort

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 minProcessingTime(self, processorTime: List[int], tasks: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Processing Time
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[]} processorTime
* @param {number[]} tasks
* @return {number}
*/
var minProcessingTime = function(processorTime, tasks) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Processing Time
* Difficulty: Medium
```

```
* Tags: array, greedy, sort
*
* 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 minProcessingTime(processorTime: number[], tasks: number[]): number
{

};
```

C# Solution:

```
/*
* Problem: Minimum Processing Time
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MinProcessingTime(IList<int> processorTime, IList<int> tasks) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Processing Time
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minProcessingTime(int* processorTime, int processorTimeSize, int* tasks,
int tasksSize) {

}
```

Go Solution:

```
// Problem: Minimum Processing Time
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minProcessingTime(processorTime []int, tasks []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minProcessingTime(processorTime: List<Int>, tasks: List<Int>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Processing Time
// Difficulty: Medium
// Tags: array, greedy, sort
```

```
//
// 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_processing_time(processor_time: Vec<i32>, tasks: Vec<i32>) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} processor_time
# @param {Integer[]} tasks
# @return {Integer}
def min_processing_time(processor_time, tasks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $processorTime
* @param Integer[] $tasks
* @return Integer
*/
function minProcessingTime($processorTime, $tasks) {

}
}
```

Dart Solution:

```
class Solution {
int minProcessingTime(List<int> processorTime, List<int> tasks) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def minProcessingTime(processorTime: List[Int], tasks: List[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_processing_time(processor_time :: [integer], tasks :: [integer]) ::
integer
def min_processing_time(processor_time, tasks) do

end
end
```

Erlang Solution:

```
-spec min_processing_time(ProcessorTime :: [integer()], Tasks :: [integer()])
-> integer().
min_processing_time(ProcessorTime, Tasks) ->
.
```

Racket Solution:

```
(define/contract (min-processing-time processorTime tasks)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```