Problem 2651: Calculate Delayed Arrival Time

Problem Information

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

Problem Description

You are given a positive integer

arrivalTime

denoting the arrival time of a train in hours, and another positive integer

delayedTime

denoting the amount of delay in hours.

Return

the time when the train will arrive at the station.

Note that the time in this problem is in 24-hours format.

Example 1:

Input:

arrivalTime = 15, delayedTime = 5

Output:

20

Explanation:

Arrival time of the train was 15:00 hours. It is delayed by 5 hours. Now it will reach at 15+5 = 20 (20:00 hours).

Example 2:

Input:

arrivalTime = 13, delayedTime = 11

Output:

0

Explanation:

Arrival time of the train was 13:00 hours. It is delayed by 11 hours. Now it will reach at 13+11=24 (Which is denoted by 00:00 in 24 hours format so return 0).

Constraints:

1 <= arrivaltime < 24

1 <= delayedTime <= 24

Code Snippets

C++:

```
class Solution {
public:
int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
};
```

Java:

```
class Solution {
public int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
}
```

Python3:

```
class Solution:
def findDelayedArrivalTime(self, arrivalTime: int, delayedTime: int) -> int:
```

Python:

```
class Solution(object):
def findDelayedArrivalTime(self, arrivalTime, delayedTime):
"""
:type arrivalTime: int
:type delayedTime: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} arrivalTime
* @param {number} delayedTime
* @return {number}
*/
var findDelayedArrivalTime = function(arrivalTime, delayedTime) {

};
```

TypeScript:

```
function findDelayedArrivalTime(arrivalTime: number, delayedTime: number):
number {

};
```

C#:

```
public class Solution {
public int FindDelayedArrivalTime(int arrivalTime, int delayedTime) {
```

```
}
}
```

C:

```
int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
```

Go:

```
func findDelayedArrivalTime(arrivalTime int, delayedTime int) int {

}
```

Kotlin:

```
class Solution {
fun findDelayedArrivalTime(arrivalTime: Int, delayedTime: Int): Int {

}
}
```

Swift:

```
class Solution {
func findDelayedArrivalTime(_ arrivalTime: Int, _ delayedTime: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_delayed_arrival_time(arrival_time: i32, delayed_time: i32) -> i32
{

}
}
```

Ruby:

```
# @param {Integer} arrival_time
# @param {Integer} delayed_time
# @return {Integer}
def find_delayed_arrival_time(arrival_time, delayed_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer $arrivalTime
* @param Integer $delayedTime
* @return Integer
*/
function findDelayedArrivalTime($arrivalTime, $delayedTime) {

}
}
```

Dart:

```
class Solution {
int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
}
```

Scala:

```
object Solution {
def findDelayedArrivalTime(arrivalTime: Int, delayedTime: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_delayed_arrival_time(arrival_time :: integer, delayed_time ::
integer) :: integer
def find_delayed_arrival_time(arrival_time, delayed_time) do
```

```
end
end
```

Erlang:

```
-spec find_delayed_arrival_time(ArrivalTime :: integer(), DelayedTime ::
integer()) -> integer().
find_delayed_arrival_time(ArrivalTime, DelayedTime) ->
.
```

Racket:

```
(define/contract (find-delayed-arrival-time arrivalTime delayedTime)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Calculate Delayed Arrival Time
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
};
```

Java Solution:

```
/**
* Problem: Calculate Delayed Arrival Time
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Calculate Delayed Arrival Time
Difficulty: Easy
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findDelayedArrivalTime(self, arrivalTime: int, delayedTime: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findDelayedArrivalTime(self, arrivalTime, delayedTime):
"""
:type arrivalTime: int
:type delayedTime: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Calculate Delayed Arrival Time
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} arrivalTime
* @param {number} delayedTime
* @return {number}
*/
var findDelayedArrivalTime = function(arrivalTime, delayedTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Calculate Delayed Arrival Time
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function findDelayedArrivalTime(arrivalTime: number, delayedTime: number):
number {

};
```

C# Solution:

```
/*
* Problem: Calculate Delayed Arrival Time
```

```
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FindDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
}
```

C Solution:

```
/*
* Problem: Calculate Delayed Arrival Time
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
```

Go Solution:

```
// Problem: Calculate Delayed Arrival Time
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func findDelayedArrivalTime(arrivalTime int, delayedTime int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun findDelayedArrivalTime(arrivalTime: Int, delayedTime: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findDelayedArrivalTime(_ arrivalTime: Int, _ delayedTime: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Calculate Delayed Arrival Time
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_delayed_arrival_time(arrival_time: i32, delayed_time: i32) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer} arrival_time
# @param {Integer} delayed_time
# @return {Integer}
```

```
def find_delayed_arrival_time(arrival_time, delayed_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $arrivalTime
* @param Integer $delayedTime
* @return Integer
*/
function findDelayedArrivalTime($arrivalTime, $delayedTime) {

}
}
```

Dart Solution:

```
class Solution {
int findDelayedArrivalTime(int arrivalTime, int delayedTime) {

}
}
```

Scala Solution:

```
object Solution {
def findDelayedArrivalTime(arrivalTime: Int, delayedTime: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_delayed_arrival_time(arrival_time :: integer, delayed_time ::
integer) :: integer
def find_delayed_arrival_time(arrival_time, delayed_time) do
```

```
end
end
```

Erlang Solution:

```
-spec find_delayed_arrival_time(ArrivalTime :: integer(), DelayedTime ::
integer()) -> integer().
find_delayed_arrival_time(ArrivalTime, DelayedTime) ->
.
```

Racket Solution:

```
(define/contract (find-delayed-arrival-time arrivalTime delayedTime)
(-> exact-integer? exact-integer? exact-integer?)
)
```