Problem 2739: Total Distance Traveled

Problem Information

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

Problem Description

A truck has two fuel tanks. You are given two integers,

mainTank

representing the fuel present in the main tank in liters and

additionalTank

representing the fuel present in the additional tank in liters.

The truck has a mileage of

10

km per liter. Whenever

5

liters of fuel get used up in the main tank, if the additional tank has at least

1

liters of fuel,

1

liters of fuel will be transferred from the additional tank to the main tank.

Return

the maximum distance which can be traveled.

Note:

Injection from the additional tank is not continuous. It happens suddenly and immediately for every 5 liters consumed.

Example 1:

Input:

mainTank = 5, additionalTank = 10

Output:

60

Explanation:

After spending 5 litre of fuel, fuel remaining is (5 - 5 + 1) = 1 litre and distance traveled is 50km. After spending another 1 litre of fuel, no fuel gets injected in the main tank and the main tank becomes empty. Total distance traveled is 60km.

Example 2:

Input:

mainTank = 1, additionalTank = 2

Output:

10

Explanation:

After spending 1 litre of fuel, the main tank becomes empty. Total distance traveled is 10km.

Constraints:

1 <= mainTank, additionalTank <= 100

Code Snippets

C++:

```
class Solution {
public:
int distanceTraveled(int mainTank, int additionalTank) {

}
};
```

Java:

```
class Solution {
public int distanceTraveled(int mainTank, int additionalTank) {

}
}
```

Python3:

```
class Solution:
def distanceTraveled(self, mainTank: int, additionalTank: int) -> int:
```

Python:

```
class Solution(object):
def distanceTraveled(self, mainTank, additionalTank):
"""
:type mainTank: int
:type additionalTank: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} mainTank
* @param {number} additionalTank
* @return {number}
*/
var distanceTraveled = function(mainTank, additionalTank) {

};
```

TypeScript:

```
function distanceTraveled(mainTank: number, additionalTank: number): number {

};
```

C#:

```
public class Solution {
public int DistanceTraveled(int mainTank, int additionalTank) {

}
}
```

C:

```
int distanceTraveled(int mainTank, int additionalTank) {

}
```

Go:

```
func distanceTraveled(mainTank int, additionalTank int) int {

}
```

Kotlin:

```
class Solution {
fun distanceTraveled(mainTank: Int, additionalTank: Int): Int {

}
}
```

Swift:

```
class Solution {
func distanceTraveled(_ mainTank: Int, _ additionalTank: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn distance_traveled(main_tank: i32, additional_tank: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} main_tank
# @param {Integer} additional_tank
# @return {Integer}
def distance_traveled(main_tank, additional_tank)

end
```

PHP:

```
class Solution {

/**
* @param Integer $mainTank
* @param Integer $additionalTank
* @return Integer
*/
function distanceTraveled($mainTank, $additionalTank) {

}
}
```

Dart:

```
class Solution {
int distanceTraveled(int mainTank, int additionalTank) {
}
}
```

Scala:

```
object Solution {
def distanceTraveled(mainTank: Int, additionalTank: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec distance_traveled(main_tank :: integer, additional_tank :: integer) ::
integer
def distance_traveled(main_tank, additional_tank) do

end
end
```

Erlang:

```
-spec distance_traveled(MainTank :: integer(), AdditionalTank :: integer())
-> integer().
distance_traveled(MainTank, AdditionalTank) ->
.
```

Racket:

```
(define/contract (distance-traveled mainTank additionalTank)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Total Distance Traveled
* 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 distanceTraveled(int mainTank, int additionalTank) {

}
};
```

Java Solution:

```
/**
* Problem: Total Distance Traveled
* 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 distanceTraveled(int mainTank, int additionalTank) {

}
}
```

Python3 Solution:

```
"""
Problem: Total Distance Traveled
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 distanceTraveled(self, mainTank: int, additionalTank: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def distanceTraveled(self, mainTank, additionalTank):
"""
:type mainTank: int
:type additionalTank: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Total Distance Traveled
* 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} mainTank
* @param {number} additionalTank
* @return {number}
*/
var distanceTraveled = function(mainTank, additionalTank) {

};
```

TypeScript Solution:

```
/**
* Problem: Total Distance Traveled
* 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 distanceTraveled(mainTank: number, additionalTank: number): number {

};
```

C# Solution:

```
/*
* Problem: Total Distance Traveled
* 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 DistanceTraveled(int mainTank, int additionalTank) {

}
}
```

C Solution:

```
/*
* Problem: Total Distance Traveled
* 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 distanceTraveled(int mainTank, int additionalTank) {

}
```

Go Solution:

```
// Problem: Total Distance Traveled
// 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 distanceTraveled(mainTank int, additionalTank int) int {

}
```

Kotlin Solution:

```
class Solution {
fun distanceTraveled(mainTank: Int, additionalTank: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func distanceTraveled(_ mainTank: Int, _ additionalTank: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Total Distance Traveled
// 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 distance_traveled(main_tank: i32, additional_tank: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} main_tank
# @param {Integer} additional_tank
# @return {Integer}
def distance_traveled(main_tank, additional_tank)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $mainTank
* @param Integer $additionalTank
* @return Integer
*/
function distanceTraveled($mainTank, $additionalTank) {

}
}
```

Dart Solution:

```
class Solution {
int distanceTraveled(int mainTank, int additionalTank) {

}

}
```

Scala Solution:

```
object Solution {
def distanceTraveled(mainTank: Int, additionalTank: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec distance_traveled(main_tank :: integer, additional_tank :: integer) ::
integer
def distance_traveled(main_tank, additional_tank) do

end
end
```

Erlang Solution:

```
-spec distance_traveled(MainTank :: integer(), AdditionalTank :: integer())
-> integer().
distance_traveled(MainTank, AdditionalTank) ->
.
```

Racket Solution:

```
(define/contract (distance-traveled mainTank additionalTank)
(-> exact-integer? exact-integer? exact-integer?)
)
```