Problem 3516: Find Closest Person

Problem Information

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

Problem Description

You are given three integers

x

,

y

, and

z

, representing the positions of three people on a number line:

x

is the position of Person 1.

y

is the position of Person 2.

z

is the position of Person 3, who does

not

move.

Both Person 1 and Person 2 move toward Person 3 at the

same

speed.

Determine which person reaches Person 3

first

:

Return 1 if Person 1 arrives first.

Return 2 if Person 2 arrives first.

Return 0 if both arrive at the

same

time.

Return the result accordingly.

Example 1:

Input:

x = 2, y = 7, z = 4

Output:

1

Explanation:

Person 1 is at position 2 and can reach Person 3 (at position 4) in 2 steps.

Person 2 is at position 7 and can reach Person 3 in 3 steps.

Since Person 1 reaches Person 3 first, the output is 1.

Example 2:

Input:

x = 2, y = 5, z = 6

Output:

2

Explanation:

Person 1 is at position 2 and can reach Person 3 (at position 6) in 4 steps.

Person 2 is at position 5 and can reach Person 3 in 1 step.

Since Person 2 reaches Person 3 first, the output is 2.

Example 3:

Input:

x = 1, y = 5, z = 3

Output:

0

Explanation:

Person 1 is at position 1 and can reach Person 3 (at position 3) in 2 steps.

Person 2 is at position 5 and can reach Person 3 in 2 steps.

Since both Person 1 and Person 2 reach Person 3 at the same time, the output is 0.

Constraints:

1 <= x, y, z <= 100

Code Snippets

C++:

```
class Solution {
public:
int findClosest(int x, int y, int z) {

}
};
```

Java:

```
class Solution {
public int findClosest(int x, int y, int z) {

}
}
```

Python3:

```
class Solution:
def findClosest(self, x: int, y: int, z: int) -> int:
```

Python:

```
class Solution(object):
def findClosest(self, x, y, z):
"""
:type x: int
:type y: int
:type z: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @param {number} z
* @return {number}
*/
var findClosest = function(x, y, z) {

};
```

TypeScript:

```
function findClosest(x: number, y: number, z: number): number {

};
```

C#:

```
public class Solution {
public int FindClosest(int x, int y, int z) {

}
}
```

C:

```
int findClosest(int x, int y, int z) {

}
```

Go:

```
func findClosest(x int, y int, z int) int {

}
```

Kotlin:

```
class Solution {
fun findClosest(x: Int, y: Int, z: Int): Int {

}
}
```

Swift:

```
class Solution {
func findClosest(_ x: Int, _ y: Int, _ z: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_closest(x: i32, y: i32, z: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} z
# @return {Integer}
def find_closest(x, y, z)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return Integer
*/
function findClosest($x, $y, $z) {
```

```
}
}
```

Dart:

```
class Solution {
int findClosest(int x, int y, int z) {

}
}
```

Scala:

```
object Solution {
def findClosest(x: Int, y: Int, z: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_closest(x :: integer, y :: integer, z :: integer) :: integer
def find_closest(x, y, z) do

end
end
```

Erlang:

```
-spec find_closest(X :: integer(), Y :: integer(), Z :: integer()) ->
integer().
find_closest(X, Y, Z) ->
.
```

Racket:

```
(define/contract (find-closest x y z)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Closest Person
* 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 findClosest(int x, int y, int z) {

}
};
```

Java Solution:

```
/**
* Problem: Find Closest Person
* 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 findClosest(int x, int y, int z) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Closest Person
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 findClosest(self, x: int, y: int, z: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findClosest(self, x, y, z):
"""
:type x: int
:type y: int
:type z: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Closest Person
* 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} x
* @param {number} y
* @param {number} z
```

```
* @return {number}
*/
var findClosest = function(x, y, z) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Closest Person
* 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 findClosest(x: number, y: number, z: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Closest Person
* 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 FindClosest(int x, int y, int z) {

}
}
```

C Solution:

```
/*
* Problem: Find Closest Person
* 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 findClosest(int x, int y, int z) {

}
```

Go Solution:

```
// Problem: Find Closest Person
// 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 findClosest(x int, y int, z int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findClosest(x: Int, y: Int, z: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findClosest(_ x: Int, _ y: Int, _ z: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Find Closest Person
// 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_closest(x: i32, y: i32, z: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} z
# @return {Integer}
def find_closest(x, y, z)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return Integer
*/
function findClosest($x, $y, $z) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int findClosest(int x, int y, int z) {

}
}
```

Scala Solution:

```
object Solution {
def findClosest(x: Int, y: Int, z: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_closest(x :: integer, y :: integer, z :: integer) :: integer
def find_closest(x, y, z) do

end
end
```

Erlang Solution:

```
-spec find_closest(X :: integer(), Y :: integer(), Z :: integer()) ->
integer().
find_closest(X, Y, Z) ->
.
```

Racket Solution:

```
(define/contract (find-closest x y z)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```