

Problem 3633: Earliest Finish Time for Land and Water Rides I

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two categories of theme park attractions:

land rides

and

water rides

Land rides

landStartTime[i]

– the earliest time the

i

th

land ride can be boarded.

landDuration[i]

– how long the

i

th

land ride lasts.

Water rides

waterStartTime[j]

– the earliest time the

j

th

water ride can be boarded.

waterDuration[j]

– how long the

j

th

water ride lasts.

A tourist must experience

exactly one

ride from

each

category, in

either order

A ride may be started at its opening time or

any later moment

If a ride is started at time

t

, it finishes at time

$t + \text{duration}$

Immediately after finishing one ride the tourist may board the other (if it is already open) or wait until it opens.

Return the

earliest possible time

at which the tourist can finish both rides.

Example 1:

Input:

`landStartTime = [2,8], landDuration = [4,1], waterStartTime = [6], waterDuration = [3]`

Output:

Explanation:

Plan A (land ride 0 → water ride 0):

Start land ride 0 at time

$$\text{landStartTime}[0] = 2$$

. Finish at

$$2 + \text{landDuration}[0] = 6$$

.

Water ride 0 opens at time

$$\text{waterStartTime}[0] = 6$$

. Start immediately at

6

, finish at

$$6 + \text{waterDuration}[0] = 9$$

.

Plan B (water ride 0 → land ride 1):

Start water ride 0 at time

$$\text{waterStartTime}[0] = 6$$

. Finish at

$$6 + \text{waterDuration}[0] = 9$$

.
Land ride 1 opens at

landStartTime[1] = 8

. Start at time

9

, finish at

9 + landDuration[1] = 10

.
Plan C (land ride 1 → water ride 0):

Start land ride 1 at time

landStartTime[1] = 8

. Finish at

8 + landDuration[1] = 9

.
Water ride 0 opened at

waterStartTime[0] = 6

. Start at time

9

, finish at

9 + waterDuration[0] = 12

Plan D (water ride 0 → land ride 0):

Start water ride 0 at time

$$\text{waterStartTime}[0] = 6$$

. Finish at

$$6 + \text{waterDuration}[0] = 9$$

Land ride 0 opened at

$$\text{landStartTime}[0] = 2$$

. Start at time

9

, finish at

$$9 + \text{landDuration}[0] = 13$$

Plan A gives the earliest finish time of 9.

Example 2:

Input:

`landStartTime = [5], landDuration = [3], waterStartTime = [1], waterDuration = [10]`

Output:

Explanation:

Plan A (water ride 0 → land ride 0):

Start water ride 0 at time

$$\text{waterStartTime}[0] = 1$$

. Finish at

$$1 + \text{waterDuration}[0] = 11$$

.

Land ride 0 opened at

$$\text{landStartTime}[0] = 5$$

. Start immediately at

$$11$$

and finish at

$$11 + \text{landDuration}[0] = 14$$

.

Plan B (land ride 0 → water ride 0):

Start land ride 0 at time

$$\text{landStartTime}[0] = 5$$

. Finish at

$5 + \text{landDuration}[0] = 8$

Water ride 0 opened at

$\text{waterStartTime}[0] = 1$

. Start immediately at

8

and finish at

$8 + \text{waterDuration}[0] = 18$

Plan A provides the earliest finish time of 14.

Constraints:

$1 \leq n, m \leq 100$

$\text{landStartTime.length} == \text{landDuration.length} == n$

$\text{waterStartTime.length} == \text{waterDuration.length} == m$

$1 \leq \text{landStartTime}[i], \text{landDuration}[i], \text{waterStartTime}[j], \text{waterDuration}[j] \leq 1000$

Code Snippets

C++:

```
class Solution {
public:
    int earliestFinishTime(vector<int>& landStartTime, vector<int>& landDuration,
```

```
vector<int>& waterStartTime, vector<int>& waterDuration) {  
}  
}  
};
```

Java:

```
class Solution {  
    public int earliestFinishTime(int[] landStartTime, int[] landDuration, int[]  
        waterStartTime, int[] waterDuration) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def earliestFinishTime(self, landStartTime: List[int], landDuration:  
        List[int], waterStartTime: List[int], waterDuration: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def earliestFinishTime(self, landStartTime, landDuration, waterStartTime,  
        waterDuration):  
        """  
        :type landStartTime: List[int]  
        :type landDuration: List[int]  
        :type waterStartTime: List[int]  
        :type waterDuration: List[int]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} landStartTime  
 * @param {number[]} landDuration  
 * @param {number[]} waterStartTime  
 * @param {number[]} waterDuration  
 * @return {number}  
 */  
var earliestFinishTime = function(landStartTime, landDuration,
```

```
waterStartTime, waterDuration) {  
  
};
```

TypeScript:

```
function earliestFinishTime(landStartTime: number[], landDuration: number[],  
waterStartTime: number[], waterDuration: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int EarliestFinishTime(int[] landStartTime, int[] landDuration, int[]  
        waterStartTime, int[] waterDuration) {  
  
    }  
}
```

C:

```
int earliestFinishTime(int* landStartTime, int landStartTimeSize, int*  
    landDuration, int landDurationSize, int* waterStartTime, int  
    waterStartTimeSize, int* waterDuration, int waterDurationSize) {  
  
}
```

Go:

```
func earliestFinishTime(landStartTime []int, landDuration []int,  
waterStartTime []int, waterDuration []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun earliestFinishTime(landStartTime: IntArray, landDuration: IntArray,  
        waterStartTime: IntArray, waterDuration: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func earliestFinishTime(_ landStartTime: [Int], _ landDuration: [Int], _  
    waterStartTime: [Int], _ waterDuration: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn earliest_finish_time(land_start_time: Vec<i32>, land_duration:  
    Vec<i32>, water_start_time: Vec<i32>, water_duration: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} land_start_time  
# @param {Integer[]} land_duration  
# @param {Integer[]} water_start_time  
# @param {Integer[]} water_duration  
# @return {Integer}  
def earliest_finish_time(land_start_time, land_duration, water_start_time,  
water_duration)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $landStartTime  
     * @param Integer[] $landDuration  
     * @param Integer[] $waterStartTime  
     * @param Integer[] $waterDuration  
     * @return Integer  
     */  
    function earliestFinishTime($landStartTime, $landDuration, $waterStartTime,  
    $waterDuration) {
```

```
}
```

```
}
```

Dart:

```
class Solution {  
    int earliestFinishTime(List<int> landStartTime, List<int> landDuration,  
    List<int> waterStartTime, List<int> waterDuration) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def earliestFinishTime(landStartTime: Array[Int], landDuration: Array[Int],  
    waterStartTime: Array[Int], waterDuration: Array[Int]): Int = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec earliest_finish_time(list(integer()), list(integer()),  
        list(integer()), list(integer())) :: integer()  
    def earliest_finish_time([land_start_time | _], [land_duration | _],  
        [water_start_time | _], [water_duration | _]) do  
  
    end  
end
```

Erlang:

```
-spec earliest_finish_time(list(integer()), list(integer()),  
    list(integer()), list(integer())) -> integer().  
earliest_finish_time([LandStartTime | _], [LandDuration | _],  
    [WaterStartTime | _], [WaterDuration | _]) ->  
    .
```

Racket:

```
(define/contract (earliest-finish-time landStartTime landDuration
waterStartTime waterDuration)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?) exact-integer?
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 earliestFinishTime(vector<int>& landStartTime, vector<int>& landDuration,
vector<int>& waterStartTime, vector<int>& waterDuration) {

}
};
```

Java Solution:

```
/**
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 earliestFinishTime(int[] landStartTime, int[] landDuration, int[]
waterStartTime, int[] waterDuration) {
}

}

```

Python3 Solution:

```

"""
Problem: Earliest Finish Time for Land and Water Rides I
Difficulty: Easy
Tags: array, greedy, sort, search

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 earliestFinishTime(self, landStartTime: List[int], landDuration:
List[int], waterStartTime: List[int], waterDuration: List[int]) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def earliestFinishTime(self, landStartTime, landDuration, waterStartTime,
waterDuration):
        """
        :type landStartTime: List[int]
        :type landDuration: List[int]
        :type waterStartTime: List[int]
        :type waterDuration: List[int]
        :rtype: int
        """

```

JavaScript Solution:

```

    /**
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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[]} landStartTime
 * @param {number[]} landDuration
 * @param {number[]} waterStartTime
 * @param {number[]} waterDuration
 * @return {number}
 */
var earliestFinishTime = function(landStartTime, landDuration,
waterStartTime, waterDuration) {

};

```

TypeScript Solution:

```

    /**
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 earliestFinishTime(landStartTime: number[], landDuration: number[],
waterStartTime: number[], waterDuration: number[]): number {

};

```

C# Solution:

```

/*
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 EarliestFinishTime(int[] landStartTime, int[] landDuration, int[]
        waterStartTime, int[] waterDuration) {

    }
}

```

C Solution:

```

/*
 * Problem: Earliest Finish Time for Land and Water Rides I
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 earliestFinishTime(int* landStartTime, int landStartTimeSize, int*
landDuration, int landDurationSize, int* waterStartTime, int
waterStartTimeSize, int* waterDuration, int waterDurationSize) {

}

```

Go Solution:

```

// Problem: Earliest Finish Time for Land and Water Rides I
// Difficulty: Easy
// Tags: array, greedy, sort, search
//
// 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 earliestFinishTime(landStartTime []int, landDuration []int,
waterStartTime []int, waterDuration []int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun earliestFinishTime(landStartTime: IntArray, landDuration: IntArray,
    waterStartTime: IntArray, waterDuration: IntArray): Int {
        return 0
    }
}

```

Swift Solution:

```

class Solution {
    func earliestFinishTime(_ landStartTime: [Int], _ landDuration: [Int], _ waterStartTime: [Int], _ waterDuration: [Int]) -> Int {
        return 0
    }
}

```

Rust Solution:

```

// Problem: Earliest Finish Time for Land and Water Rides I
// Difficulty: Easy
// Tags: array, greedy, sort, search
//
// 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 earliest_finish_time(land_start_time: Vec<i32>, land_duration:
    Vec<i32>, water_start_time: Vec<i32>, water_duration: Vec<i32>) -> i32 {
        return 0
    }
}

```

```
}
```

Ruby Solution:

```
# @param {Integer[]} land_start_time
# @param {Integer[]} land_duration
# @param {Integer[]} water_start_time
# @param {Integer[]} water_duration
# @return {Integer}

def earliest_finish_time(land_start_time, land_duration, water_start_time,
water_duration)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $landStartTime
     * @param Integer[] $landDuration
     * @param Integer[] $waterStartTime
     * @param Integer[] $waterDuration
     * @return Integer
     */

    function earliestFinishTime($landStartTime, $landDuration, $waterStartTime,
$waterDuration) {

    }
}
```

Dart Solution:

```
class Solution {
int earliestFinishTime(List<int> landStartTime, List<int> landDuration,
List<int> waterStartTime, List<int> waterDuration) {

}
```

Scala Solution:

```

object Solution {
    def earliestFinishTime(landStartTime: Array[Int], landDuration: Array[Int],
    waterStartTime: Array[Int], waterDuration: Array[Int]): Int = {
        }
    }
}

```

Elixir Solution:

```

defmodule Solution do
  @spec earliest_finish_time(list(integer()), list(integer()),
    list(integer()), list(integer())) :: integer()
  def earliest_finish_time([land_start_time | _], [land_duration | _], [water_start_time | _], [water_duration | _]) do
    end
  end
end

```

Erlang Solution:

```

-spec earliest_finish_time(list(integer()), list(integer()),
    list(integer()), list(integer())) -> integer().
earliest_finish_time([LandStartTime | _], [LandDuration | _], [WaterStartTime | _], [WaterDuration | _]) ->
  .
.
```

Racket Solution:

```

(define/contract (earliest-finish-time landStartTime landDuration
    waterStartTime waterDuration)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
    (listof exact-integer?) exact-integer?))
)
```