# Problem 2136: Earliest Possible Day of Full Bloom

## Problem Information

**Difficulty:** Hard
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You have

n

flower seeds. Every seed must be planted first before it can begin to grow, then bloom. Planting a seed takes time and so does the growth of a seed. You are given two

0-indexed

integer arrays

plantTime

and

growTime

, of length

n

each:

plantTime[i]

is the number of

full days

it takes you to

plant

the

i

th

seed. Every day, you can work on planting exactly one seed. You

do not

have to work on planting the same seed on consecutive days, but the planting of a seed is not complete

until

you have worked

plantTime[i]

days on planting it in total.

growTime[i]

is the number of

full days

it takes the

i

th

seed to grow after being completely planted.

After

the last day of its growth, the flower

blooms

and stays bloomed forever.

From the beginning of day

0

, you can plant the seeds in

any

order.

Return

the

earliest

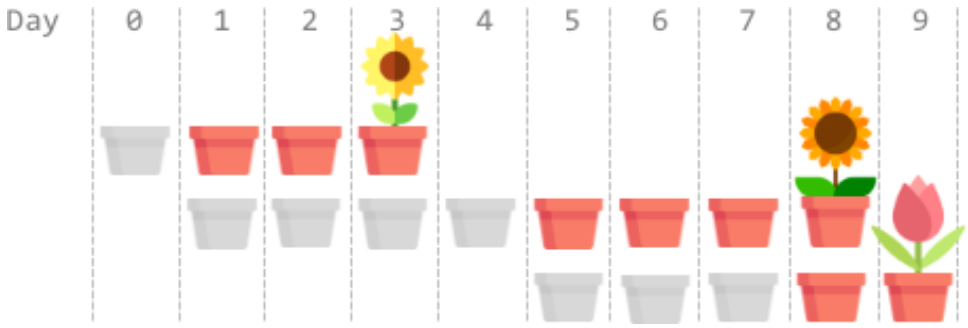
possible day where

all

seeds are blooming

.

Example 1:




Input:

plantTime = [1,4,3], growTime = [2,3,1]

Output:

9

Explanation:

The grayed out pots represent planting days, colored pots represent growing days, and the flower represents the day it blooms. One optimal way is: On day 0, plant the 0

th

seed. The seed grows for 2 full days and blooms on day 3. On days 1, 2, 3, and 4, plant the 1

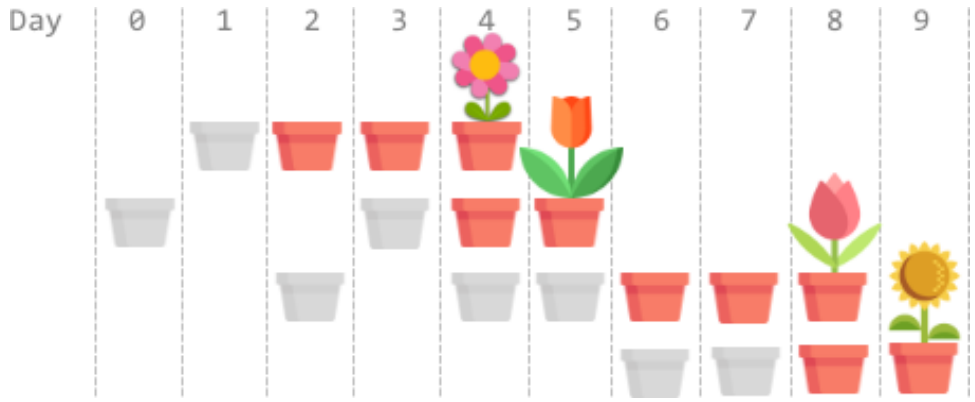
st

seed. The seed grows for 3 full days and blooms on day 8. On days 5, 6, and 7, plant the 2

nd

seed. The seed grows for 1 full day and blooms on day 9. Thus, on day 9, all the seeds are blooming.

Example 2:




Input:

plantTime = [1,2,3,2], growTime = [2,1,2,1]

Output:

9

Explanation:

The grayed out pots represent planting days, colored pots represent growing days, and the flower represents the day it blooms. One optimal way is: On day 1, plant the 0

th

seed. The seed grows for 2 full days and blooms on day 4. On days 0 and 3, plant the 1

st

seed. The seed grows for 1 full day and blooms on day 5. On days 2, 4, and 5, plant the 2

nd

seed. The seed grows for 2 full days and blooms on day 8. On days 6 and 7, plant the 3

rd

seed. The seed grows for 1 full day and blooms on day 9. Thus, on day 9, all the seeds are blooming.

Example 3:

Input:

plantTime = [1], growTime = [1]

Output:

2

Explanation:

On day 0, plant the 0

th

seed. The seed grows for 1 full day and blooms on day 2. Thus, on day 2, all the seeds are blooming.

Constraints:

n == plantTime.length == growTime.length

1 <= n <= 10

5

1 <= plantTime[i], growTime[i] <= 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
int earliestFullBloom(vector<int>& plantTime, vector<int>& growTime) {
```

```
}
};
```

**Java:**

```
class Solution {
public int earliestFullBloom(int[] plantTime, int[] growTime) {

}
}
```

**Python3:**

```
class Solution:
def earliestFullBloom(self, plantTime: List[int], growTime: List[int]) ->
int:
```

**Python:**

```
class Solution(object):
def earliestFullBloom(self, plantTime, growTime):
"""
:type plantTime: List[int]
:type growTime: List[int]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} plantTime
* @param {number[]} growTime
* @return {number}
*/
var earliestFullBloom = function(plantTime, growTime) {

};
```

**TypeScript:**

```
function earliestFullBloom(plantTime: number[], growTime: number[]): number {

};
```

**C#:**

```
public class Solution {
public int EarliestFullBloom(int[] plantTime, int[] growTime) {

}
}
```

**C:**

```
int earliestFullBloom(int* plantTime, int plantTimeSize, int* growTime, int
growTimeSize) {

}
```

**Go:**

```
func earliestFullBloom(plantTime []int, growTime []int) int {

}
```

**Kotlin:**

```
class Solution {
fun earliestFullBloom(plantTime: IntArray, growTime: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func earliestFullBloom(_ plantTime: [Int], _ growTime: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn earliest_full_bloom(plant_time: Vec<i32>, grow_time: Vec<i32>) -> i32
{

}
}
```

**Ruby:**

```
# @param {Integer[]} plant_time
# @param {Integer[]} grow_time
# @return {Integer}
def earliest_full_bloom(plant_time, grow_time)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $plantTime
* @param Integer[] $growTime
* @return Integer
*/
function earliestFullBloom($plantTime, $growTime) {

}
}
```

**Dart:**

```
class Solution {
int earliestFullBloom(List<int> plantTime, List<int> growTime) {

}
}
```

**Scala:**

```
object Solution {
def earliestFullBloom(plantTime: Array[Int], growTime: Array[Int]): Int = {
```

```
}
}
```

**Elixir:**

```
defmodule Solution do
@spec earliest_full_bloom(plant_time :: [integer], grow_time :: [integer]) ::
integer
def earliest_full_bloom(plant_time, grow_time) do

end
end
```

**Erlang:**

```
-spec earliest_full_bloom(PlantTime :: [integer()], GrowTime :: [integer()])
-> integer().
earliest_full_bloom(PlantTime, GrowTime) ->
.
```

**Racket:**

```
(define/contract (earliest-full-bloom plantTime growTime)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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 earliestFullBloom(vector<int>& plantTime, vector<int>& growTime) {

}
};
```

### Java Solution:

```
/**
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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 earliestFullBloom(int[] plantTime, int[] growTime) {

}
}
```

### Python3 Solution:

```
"""
Problem: Earliest Possible Day of Full Bloom
Difficulty: Hard
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 earliestFullBloom(self, plantTime: List[int], growTime: List[int]) ->
int:
# TODO: Implement optimized solution
```

```
pass
```

### Python Solution:

```
class Solution(object):
def earliestFullBloom(self, plantTime, growTime):
"""
:type plantTime: List[int]
:type growTime: List[int]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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[]} plantTime
* @param {number[]} growTime
* @return {number}
*/
var earliestFullBloom = function(plantTime, growTime) {

};
```

### TypeScript Solution:

```
/**
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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 earliestFullBloom(plantTime: number[], growTime: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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 EarliestFullBloom(int[] plantTime, int[] growTime) {

}
}
```

### C Solution:

```
/*
* Problem: Earliest Possible Day of Full Bloom
* Difficulty: Hard
* 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 earliestFullBloom(int* plantTime, int plantTimeSize, int* growTime, int
growTimeSize) {
```

```
}
```

### Go Solution:

```
// Problem: Earliest Possible Day of Full Bloom
// Difficulty: Hard
// 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 earliestFullBloom(plantTime []int, growTime []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun earliestFullBloom(plantTime: IntArray, growTime: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func earliestFullBloom(_ plantTime: [Int], _ growTime: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Earliest Possible Day of Full Bloom
// Difficulty: Hard
// 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 earliest_full_bloom(plant_time: Vec<i32>, grow_time: Vec<i32>) -> i32
{

}
}
```

### Ruby Solution:

```
# @param {Integer[]} plant_time
# @param {Integer[]} grow_time
# @return {Integer}
def earliest_full_bloom(plant_time, grow_time)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $plantTime
* @param Integer[] $growTime
* @return Integer
*/
function earliestFullBloom($plantTime, $growTime) {

}
}
```

### Dart Solution:

```
class Solution {
int earliestFullBloom(List<int> plantTime, List<int> growTime) {

}
}
```

### Scala Solution:

```
object Solution {
def earliestFullBloom(plantTime: Array[Int], growTime: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec earliest_full_bloom(plant_time :: [integer], grow_time :: [integer]) ::
integer
def earliest_full_bloom(plant_time, grow_time) do

end
end
```

### Erlang Solution:

```
-spec earliest_full_bloom(PlantTime :: [integer()], GrowTime :: [integer()])
-> integer().
earliest_full_bloom(PlantTime, GrowTime) ->
.
```

### Racket Solution:

```
(define/contract (earliest-full-bloom plantTime growTime)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```