# Problem 1276: Number of Burgers with No Waste of Ingredients

## Problem Information

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

## Problem Description

Given two integers

tomatoSlices

and

cheeseSlices

. The ingredients of different burgers are as follows:

Jumbo Burger:

4

tomato slices and

1

cheese slice.

Small Burger:

2

Tomato slices and

1

cheese slice.

Return

[total_jumbo, total_small]

so that the number of remaining

tomatoSlices

equal to

0

and the number of remaining

cheeseSlices

equal to

0

. If it is not possible to make the remaining

tomatoSlices

and

cheeseSlices

equal to

0

return

[]

.

Example 1:

Input:

tomatoSlices = 16, cheeseSlices = 7

Output:

[1,6]

Explantion:

To make one jumbo burger and 6 small burgers we need 4*1 + 2*6 = 16 tomato and 1 + 6 = 7 cheese. There will be no remaining ingredients.

Example 2:

Input:

tomatoSlices = 17, cheeseSlices = 4

Output:

[]

Explantion:

There will be no way to use all ingredients to make small and jumbo burgers.

Example 3:

Input:

tomatoSlices = 4, cheeseSlices = 17

Output:

[]

Explantion:

Making 1 jumbo burger there will be 16 cheese remaining and making 2 small burgers there will be 15 cheese remaining.

Constraints:

0 <= tomatoSlices, cheeseSlices <= 10

7

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
};
```

**Java:**

```
class Solution {
public List<Integer> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
}
```

**Python3:**

```
class Solution:
def numOfBurgers(self, tomatoSlices: int, cheeseSlices: int) -> List[int]:
```

**Python:**

```
class Solution(object):
def numOfBurgers(self, tomatoSlices, cheeseSlices):
"""
:type tomatoSlices: int
:type cheeseSlices: int
:rtype: List[int]
"""
```

**JavaScript:**

```
/**
* @param {number} tomatoSlices
* @param {number} cheeseSlices
* @return {number[]}
*/
var numOfBurgers = function(tomatoSlices, cheeseSlices) {

};
```

**TypeScript:**

```
function numOfBurgers(tomatoSlices: number, cheeseSlices: number): number[] {

};
```

**C#:**

```
public class Solution {
public IList<int> NumOfBurgers(int tomatoSlices, int cheeseSlices) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numOfBurgers(int tomatoSlices, int cheeseSlices, int* returnSize) {

}
```

**Go:**

```
func numOfBurgers(tomatoSlices int, cheeseSlices int) []int {

}
```

**Kotlin:**

```
class Solution {
fun numOfBurgers(tomatoSlices: Int, cheeseSlices: Int): List<Int> {

}
}
```

**Swift:**

```
class Solution {
func numOfBurgers(_ tomatoSlices: Int, _ cheeseSlices: Int) -> [Int] {

}
}
```

**Rust:**

```
impl Solution {
pub fn num_of_burgers(tomato_slices: i32, cheese_slices: i32) -> Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer} tomato_slices
# @param {Integer} cheese_slices
# @return {Integer[]}
def num_of_burgers(tomato_slices, cheese_slices)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $tomatoSlices
```

```
* @param Integer $cheeseSlices
* @return Integer[]
*/
function numOfBurgers($tomatoSlices, $cheeseSlices) {

}
}
```

**Dart:**

```
class Solution {
List<int> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
}
```

**Scala:**

```
object Solution {
def numOfBurgers(tomatoSlices: Int, cheeseSlices: Int): List[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec num_of_burgers(tomato_slices :: integer, cheese_slices :: integer) ::
[integer]
def num_of_burgers(tomato_slices, cheese_slices) do

end
end
```

**Erlang:**

```
-spec num_of_burgers(TomatoSlices :: integer(), CheeseSlices :: integer()) ->
[integer()].
num_of_burgers(TomatoSlices, CheeseSlices) ->
.
```

**Racket:**

```
(define/contract (num-of-burgers tomatoSlices cheeseSlices)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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:
vector<int> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
};
```

### Java Solution:

```
/**
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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 List<Integer> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Number of Burgers with No Waste of Ingredients
Difficulty: Medium
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 numOfBurgers(self, tomatoSlices: int, cheeseSlices: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def numOfBurgers(self, tomatoSlices, cheeseSlices):
"""
:type tomatoSlices: int
:type cheeseSlices: int
:rtype: List[int]
"""
```

### JavaScript Solution:

```
/**
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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} tomatoSlices
* @param {number} cheeseSlices
* @return {number[]}
*/
var numOfBurgers = function(tomatoSlices, cheeseSlices) {

};
```

### TypeScript Solution:

```
/**
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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 numOfBurgers(tomatoSlices: number, cheeseSlices: number): number[] {

};
```

### C# Solution:

```
/*
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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 IList<int> NumOfBurgers(int tomatoSlices, int cheeseSlices) {

}
```

```
}
```

### C Solution:

```
/*
* Problem: Number of Burgers with No Waste of Ingredients
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numOfBurgers(int tomatoSlices, int cheeseSlices, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Number of Burgers with No Waste of Ingredients
// Difficulty: Medium
// 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 numOfBurgers(tomatoSlices int, cheeseSlices int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun numOfBurgers(tomatoSlices: Int, cheeseSlices: Int): List<Int> {

}
```

```
}
```

### Swift Solution:

```
class Solution {
func numOfBurgers(_ tomatoSlices: Int, _ cheeseSlices: Int) -> [Int] {

}
}
```

### Rust Solution:

```
// Problem: Number of Burgers with No Waste of Ingredients
// Difficulty: Medium
// 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 num_of_burgers(tomato_slices: i32, cheese_slices: i32) -> Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer} tomato_slices
# @param {Integer} cheese_slices
# @return {Integer[]}
def num_of_burgers(tomato_slices, cheese_slices)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $tomatoSlices
```

```
* @param Integer $cheeseSlices
* @return Integer[]
*/
function numOfBurgers($tomatoSlices, $cheeseSlices) {

}
}
```

**Dart Solution:**

```
class Solution {
List<int> numOfBurgers(int tomatoSlices, int cheeseSlices) {

}
}
```

**Scala Solution:**

```
object Solution {
def numOfBurgers(tomatoSlices: Int, cheeseSlices: Int): List[Int] = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec num_of_burgers(tomato_slices :: integer, cheese_slices :: integer) ::
[integer]
def num_of_burgers(tomato_slices, cheese_slices) do

end
end
```

**Erlang Solution:**

```
-spec num_of_burgers(TomatoSlices :: integer(), CheeseSlices :: integer()) ->
[integer()].
num_of_burgers(TomatoSlices, CheeseSlices) ->
.
```

**Racket Solution:**

```
(define/contract (num-of-burgers tomatoSlices cheeseSlices)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```