Problem 1447: Simplified Fractions

Problem Information

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

Problem Description

Given an integer

n

, return

a list of all

simplified

fractions between

0

and

1

(exclusive) such that the denominator is less-than-or-equal-to

n

. You can return the answer in

any order

.

Example 1:

Input:

n = 2

Output:

["1/2"]

Explanation:

"1/2" is the only unique fraction with a denominator less-than-or-equal-to 2.

Example 2:

Input:

n = 3

Output:

["1/2","1/3","2/3"]

Example 3:

Input:

n = 4

Output:

["1/2","1/3","1/4","2/3","3/4"]

Explanation:

"2/4" is not a simplified fraction because it can be simplified to "1/2".

Constraints:

1 <= n <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<string> simplifiedFractions(int n) {

}
};
```

Java:

```
class Solution {
public List<String> simplifiedFractions(int n) {

}
}
```

Python3:

```
class Solution:
def simplifiedFractions(self, n: int) -> List[str]:
```

Python:

```
class Solution(object):
def simplifiedFractions(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {string[]}
*/
var simplifiedFractions = function(n) {

};
```

TypeScript:

```
function simplifiedFractions(n: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> SimplifiedFractions(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** simplifiedFractions(int n, int* returnSize) {

}
```

Go:

```
func simplifiedFractions(n int) []string {

}
```

Kotlin:

```
class Solution {
fun simplifiedFractions(n: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func simplifiedFractions(_ n: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn simplified_fractions(n: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String[]}
def simplified_fractions(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function simplifiedFractions($n) {

}
}
```

Dart:

```
class Solution {
List<String> simplifiedFractions(int n) {

}

}
```

Scala:

```
object Solution {
def simplifiedFractions(n: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec simplified_fractions(n :: integer) :: [String.t]
def simplified_fractions(n) do

end
end
```

Erlang:

```
-spec simplified_fractions(N :: integer()) -> [unicode:unicode_binary()].
simplified_fractions(N) ->
.
```

Racket:

```
(define/contract (simplified-fractions n)
(-> exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> simplifiedFractions(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> simplifiedFractions(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Simplified Fractions
Difficulty: Medium
Tags: string, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""
```

```
class Solution:
def simplifiedFractions(self, n: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def simplifiedFractions(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {string[]}
*/
var simplifiedFractions = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function simplifiedFractions(n: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<string> SimplifiedFractions(int n) {

}
}
```

C Solution:

```
/*
* Problem: Simplified Fractions
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
char** simplifiedFractions(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Simplified Fractions
// Difficulty: Medium
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func simplifiedFractions(n int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun simplifiedFractions(n: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func simplifiedFractions(_ n: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Simplified Fractions
// Difficulty: Medium
// Tags: string, math
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn simplified_fractions(n: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String[]}
def simplified_fractions(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function simplifiedFractions($n) {

}
}
```

Dart Solution:

```
class Solution {
List<String> simplifiedFractions(int n) {

}
}
```

Scala Solution:

```
object Solution {
def simplifiedFractions(n: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec simplified_fractions(n :: integer) :: [String.t]
def simplified_fractions(n) do

end
end
```

Erlang Solution:

```
-spec simplified_fractions(N :: integer()) -> [unicode:unicode_binary()].
simplified_fractions(N) ->
.
```

Racket Solution:

```
(define/contract (simplified-fractions n)
(-> exact-integer? (listof string?))
)
```