Problem 1362: Closest Divisors

Problem Information

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

Problem Description

Given an integer

num

, find the closest two integers in absolute difference whose product equals

num + 1

or

num + 2

.

Return the two integers in any order.

Example 1:

Input:

num = 8

Output:

[3,3]

Explanation:

For num + 1 = 9, the closest divisors are 3 & 3, for num + 2 = 10, the closest divisors are 2 & 5, hence 3 & 3 is chosen.

Example 2:

Input:

num = 123

Output:

[5,25]

Example 3:

Input:

num = 999

Output:

[40,25]

Constraints:

1 <= num <= 10^9

Code Snippets

C++:

```
class Solution {
public:
vector<int> closestDivisors(int num) {

}
};
```

Java:

```
class Solution {
public int[] closestDivisors(int num) {

}
}
```

Python3:

```
class Solution:
def closestDivisors(self, num: int) -> List[int]:
```

Python:

```
class Solution(object):
def closestDivisors(self, num):
"""
:type num: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} num
* @return {number[]}
*/
var closestDivisors = function(num) {

};
```

TypeScript:

```
function closestDivisors(num: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ClosestDivisors(int num) {
```

```
}
}
```

C:

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

}
```

Go:

```
func closestDivisors(num int) []int {

}
```

Kotlin:

```
class Solution {
fun closestDivisors(num: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func closestDivisors(_ num: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn closest_divisors(num: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer[]}
def closest_divisors(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer[]
*/
function closestDivisors($num) {

}
}
```

Dart:

```
class Solution {
List<int> closestDivisors(int num) {

}
}
```

Scala:

```
object Solution {
def closestDivisors(num: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec closest_divisors(num :: integer) :: [integer]
def closest_divisors(num) do
```

```
end
end
```

Erlang:

```
-spec closest_divisors(Num :: integer()) -> [integer()].
closest_divisors(Num) ->
.
```

Racket:

```
(define/contract (closest-divisors num)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Divisors
* 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> closestDivisors(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Divisors
```

```
* 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 int[] closestDivisors(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Divisors
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 closestDivisors(self, num: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def closestDivisors(self, num):
"""
:type num: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Closest Divisors
* 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} num
* @return {number[]}
*/
var closestDivisors = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Divisors
* 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 closestDivisors(num: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Closest Divisors
* 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 int[] ClosestDivisors(int num) {

}
}
```

C Solution:

```
/*
* Problem: Closest Divisors
* 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* closestDivisors(int num, int* returnSize) {

}
```

Go Solution:

```
// Problem: Closest Divisors
// 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 closestDivisors(num int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun closestDivisors(num: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func closestDivisors(_ num: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Closest Divisors
// 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 closest_divisors(num: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer[]}
def closest_divisors(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @return Integer[]
*/
function closestDivisors($num) {

}
}
```

Dart Solution:

```
class Solution {
List<int> closestDivisors(int num) {

}
}
```

Scala Solution:

```
object Solution {
def closestDivisors(num: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec closest_divisors(num :: integer) :: [integer]
def closest_divisors(num) do

end
end
```

Erlang Solution:

```
-spec closest_divisors(Num :: integer()) -> [integer()].
closest_divisors(Num) ->
```

```
.
```

Racket Solution:

```
(define/contract (closest-divisors num)
(-> exact-integer? (listof exact-integer?))
)
```