Problem 2481: Minimum Cuts to Divide a Circle

Problem Information

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

Problem Description

A

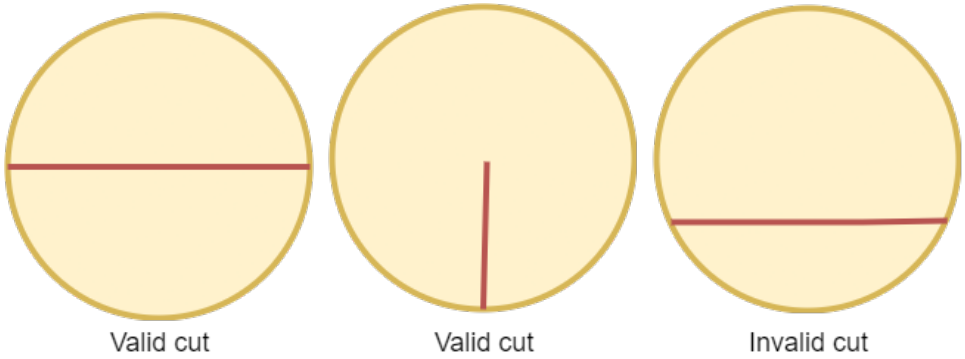
valid cut

in a circle can be:

A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or

A cut that is represented by a straight line that touches one point on the edge of the circle and its center.

Some valid and invalid cuts are shown in the figures below.



Given the integer

n

, return

the

minimum

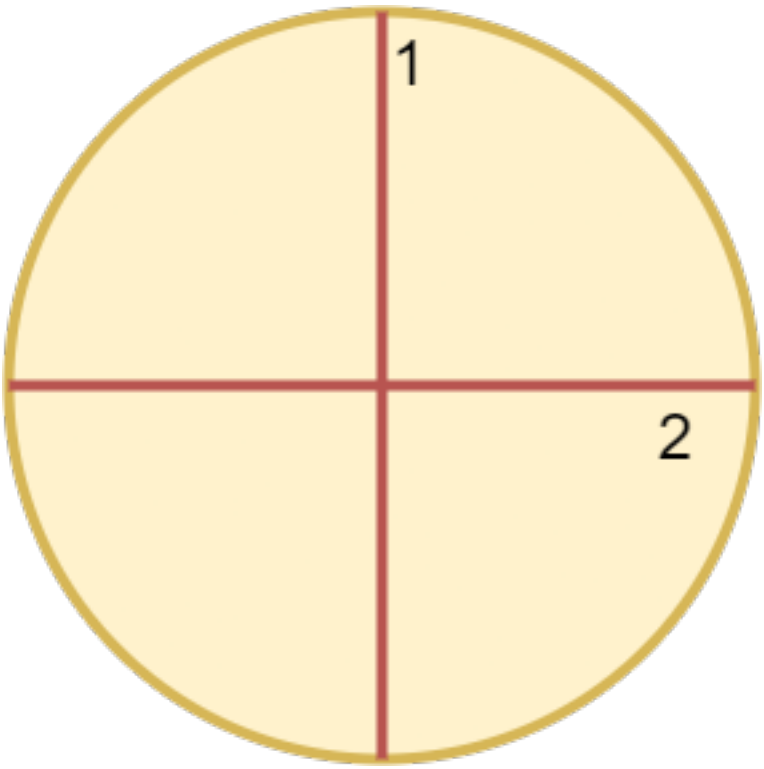
number of cuts needed to divide a circle into

n

equal slices

.

Example 1:



Input:

n = 4

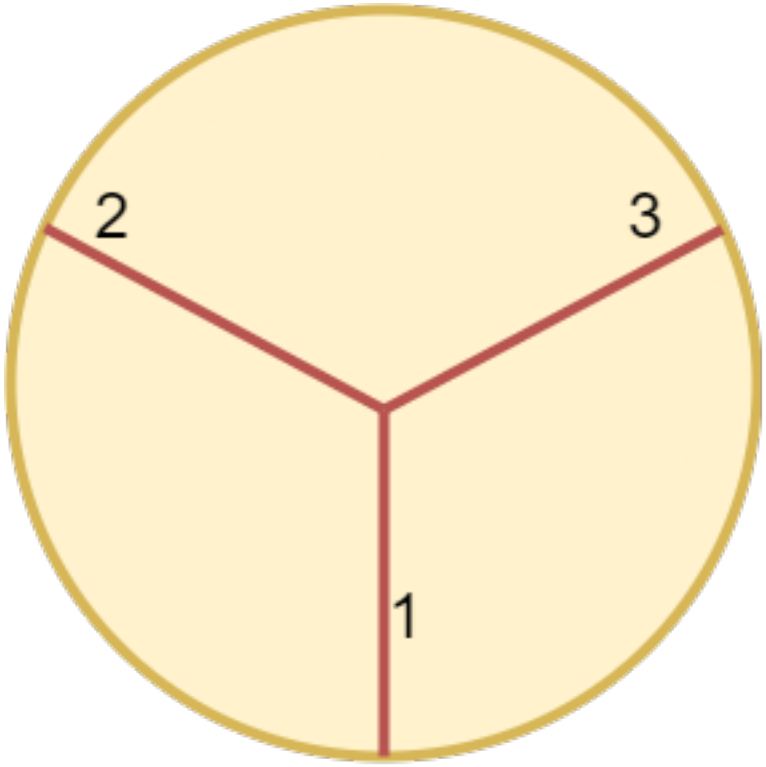
Output:

2

Explanation:

The above figure shows how cutting the circle twice through the middle divides it into 4 equal slices.

Example 2:



Input:

n = 3

Output:

3

Explanation:

At least 3 cuts are needed to divide the circle into 3 equal slices. It can be shown that less than 3 cuts cannot result in 3 slices of equal size and shape. Also note that the first cut will not divide the circle into distinct parts.

Constraints:

1 <= n <= 100

Code Snippets

C++:

```
class Solution {
public:
int numberOfCuts(int n) {

}
};
```

Java:

```
class Solution {
public int numberOfCuts(int n) {

}
}
```

Python3:

```
class Solution:
def numberOfCuts(self, n: int) -> int:
```

Python:

```
class Solution(object):
def numberOfCuts(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
```

```
* @return {number}
*/
var numberOfCuts = function(n) {

};
```

TypeScript:

```
function numberOfCuts(n: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfCuts(int n) {

}
}
```

C:

```
int numberOfCuts(int n) {

}
```

Go:

```
func numberOfCuts(n int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfCuts(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfCuts(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_cuts(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def number_of_cuts(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function numberOfCuts($n) {

}
}
```

Dart:

```
class Solution {
int numberOfCuts(int n) {

}
}
```

Scala:

```
object Solution {
def numberOfCuts(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_cuts(n :: integer) :: integer
def number_of_cuts(n) do

end
end
```

Erlang:

```
-spec number_of_cuts(N :: integer()) -> integer().
number_of_cuts(N) ->
.
```

Racket:

```
(define/contract (number-of-cuts n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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 numberOfCuts(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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 numberOfCuts(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cuts to Divide a Circle
Difficulty: Easy
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 numberOfCuts(self, n: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def numberOfCuts(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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} n
* @return {number}
*/
var numberOfCuts = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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 numberOfCuts(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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 NumberOfCuts(int n) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cuts to Divide a Circle
* Difficulty: Easy
* 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
*/

int numberOfCuts(int n) {

}
```

Go Solution:

```
// Problem: Minimum Cuts to Divide a Circle
// Difficulty: Easy
// 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 numberOfCuts(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfCuts(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfCuts(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cuts to Divide a Circle
// Difficulty: Easy
// 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 number_of_cuts(n: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def number_of_cuts(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function numberOfCuts($n) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfCuts(int n) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfCuts(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_cuts(n :: integer) :: integer
def number_of_cuts(n) do

end
end
```

Erlang Solution:

```
-spec number_of_cuts(N :: integer()) -> integer().
number_of_cuts(N) ->
.
```

Racket Solution:

```
(define/contract (number-of-cuts n)
(-> exact-integer? exact-integer?)
)
```