Problem 1688: Count of Matches in Tournament

Problem Information

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

Problem Description

You are given an integer

n

, the number of teams in a tournament that has strange rules:

If the current number of teams is

even

, each team gets paired with another team. A total of

n / 2

matches are played, and

n / 2

teams advance to the next round.

If the current number of teams is

odd

, one team randomly advances in the tournament, and the rest gets paired. A total of

(n - 1) / 2

matches are played, and

(n - 1) / 2 + 1

teams advance to the next round.

Return

the number of matches played in the tournament until a winner is decided.

Example 1:

Input:

n = 7

Output:

6

Explanation:

Details of the tournament: - 1st Round: Teams = 7, Matches = 3, and 4 teams advance. - 2nd Round: Teams = 4, Matches = 2, and 2 teams advance. - 3rd Round: Teams = 2, Matches = 1, and 1 team is declared the winner. Total number of matches = 3 + 2 + 1 = 6.

Example 2:

Input:

n = 14

Output:

13

Explanation:

Details of the tournament: - 1st Round: Teams = 14, Matches = 7, and 7 teams advance. - 2nd Round: Teams = 7, Matches = 3, and 4 teams advance. - 3rd Round: Teams = 4, Matches = 2, and 2 teams advance. - 4th Round: Teams = 2, Matches = 1, and 1 team is declared the winner. Total number of matches = 7 + 3 + 2 + 1 = 13.

Constraints:

1 <= n <= 200

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```


int numberOfMatches(int n){

}
```

Go:

```
func numberOfMatches(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Count of Matches in Tournament
* 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 numberOfMatches(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Count of Matches in Tournament
* 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 numberOfMatches(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count of Matches in Tournament
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 numberOfMatches(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count of Matches in Tournament
* 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 numberOfMatches = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count of Matches in Tournament
* 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 numberOfMatches(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Count of Matches in Tournament
* 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 NumberOfMatches(int n) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Count of Matches in Tournament
* 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 numberOfMatches(int n){

}
```

Go Solution:

```
// Problem: Count of Matches in Tournament
// 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 numberOfMatches(n int) int {

}
```

Kotlin Solution:

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

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count of Matches in Tournament
// 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_matches(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
```

```
* @return Integer
*/
function numberOfMatches($n) {

}
}
```

Scala Solution:

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

}
}
```