Problem 544: Output Contest Matches

Problem Information

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

Problem Description

During the NBA playoffs, we always set the rather strong team to play with the rather weak team, like making the rank

1

team play with the rank

n

th

team, which is a good strategy to make the contest more interesting.

Given

n

teams, return

their final contest matches in the form of a string

.

The

n

teams are labeled from

1

to

n

, which represents their initial rank (i.e., Rank

1

is the strongest team and Rank

n

is the weakest team).

We will use parentheses

'('

, and

')'

and commas

','

to represent the contest team pairing. We use the parentheses for pairing and the commas for partition. During the pairing process in each round, you always need to follow the strategy of making the rather strong one pair with the rather weak one.

Example 1:

Input:

n = 4

Output:

"((1,4),(2,3))"

Explanation:

In the first round, we pair the team 1 and 4, the teams 2 and 3 together, as we need to make the strong team and weak team together. And we got (1, 4),(2, 3). In the second round, the winners of (1, 4) and (2, 3) need to play again to generate the final winner, so you need to add the paratheses outside them. And we got the final answer ((1,4),(2,3)).

Example 2:

Input:

n = 8

Output:

"(((1,8),(4,5)),((2,7),(3,6)))"

Explanation:

First round: (1, 8),(2, 7),(3, 6),(4, 5) Second round: ((1, 8),(4, 5)),((2, 7),(3, 6)) Third round: (((1, 8),(4, 5)),((2, 7),(3, 6))) Since the third round will generate the final winner, you need to output the answer (((1,8),(4,5)),((2,7),(3,6))).

Constraints:

n == 2

x

where

x

in in the range

[1, 12]

.

Code Snippets

C++:

```
class Solution {
public:
string findContestMatch(int n) {

}
};
```

Java:

```
class Solution {
public String findContestMatch(int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findContestMatch(n: number): string {

};
```

C#:

```
public class Solution {
public string FindContestMatch(int n) {

}
}
```

C:

```
char* findContestMatch(int n) {

}
```

Go:

```
func findContestMatch(n int) string {

}
```

Kotlin:

```
class Solution {
fun findContestMatch(n: Int): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String findContestMatch(int n) {

}
}
```

Scala:

```
object Solution {
def findContestMatch(n: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_contest_match(n :: integer) :: String.t
def find_contest_match(n) do

end
end
```

Erlang:

```
-spec find_contest_match(N :: integer()) -> unicode:unicode_binary().
find_contest_match(N) ->
.
```

Racket:

```
(define/contract (find-contest-match n)
(-> exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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:
string findContestMatch(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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 String findContestMatch(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Output Contest Matches
Difficulty: Medium
Tags: string

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 findContestMatch(self, n: int) -> str:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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 findContestMatch = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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 findContestMatch(n: number): string {

};
```

C# Solution:

```
/*
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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 string FindContestMatch(int n) {

}
}
```

C Solution:

```
/*
* Problem: Output Contest Matches
* Difficulty: Medium
* Tags: string
*
* 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
*/

char* findContestMatch(int n) {

}
```

Go Solution:

```
// Problem: Output Contest Matches
// Difficulty: Medium
// Tags: string
//
// 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 findContestMatch(n int) string {

}
```

Kotlin Solution:

```
class Solution {
fun findContestMatch(n: Int): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Output Contest Matches
// Difficulty: Medium
// Tags: string
//
// 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 find_contest_match(n: i32) -> String {

}
}
```

Ruby Solution:

```ruby
# @param {Integer} n
# @return {String}
def find_contest_match(n)

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

```dart
class Solution {
String findContestMatch(int n) {

}
}
```

Scala Solution:

```scala
object Solution {
def findContestMatch(n: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_contest_match(n :: integer) :: String.t
def find_contest_match(n) do

end
end
```

Erlang Solution:

```
-spec find_contest_match(N :: integer()) -> unicode:unicode_binary().
find_contest_match(N) ->
.
```

Racket Solution:

```
(define/contract (find-contest-match n)
(-> exact-integer? string?)
)
```