# Problem 2417: Closest Fair Integer

## Problem Information

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

## Problem Description

You are given a

positive

integer

n

.

We call an integer

k

fair if the number of

even

digits in

k

is equal to the number of

odd

digits in it.

Return

the

smallest

fair integer that is

greater than or equal

to

n

.

Example 1:

Input:

n = 2

Output:

10

Explanation:

The smallest fair integer that is greater than or equal to 2 is 10. 10 is fair because it has an equal number of even and odd digits (one odd digit and one even digit).

Example 2:

Input:

n = 403

Output:

1001

Explanation:

The smallest fair integer that is greater than or equal to 403 is 1001. 1001 is fair because it has an equal number of even and odd digits (two odd digits and two even digits).

Constraints:

1 <= n <= 10

9

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

```
int closestFair(int n) {

}
```

**Go:**

```
func closestFair(n int) int {

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
```

```
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (closest-fair n)
(-> exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Closest Fair Integer
* 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 closestFair(int n) {

}
};
```

### Java Solution:

```
/**
* Problem: Closest Fair Integer
* 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 closestFair(int n) {

}
}
```

### Python3 Solution:

```
"""
Problem: Closest Fair Integer
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 closestFair(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Closest Fair Integer
* 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} n
* @return {number}
*/
var closestFair = function(n) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Closest Fair Integer
* 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 closestFair(n: number): number {

};
```

### C# Solution:

```
/*
* Problem: Closest Fair Integer
* 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 ClosestFair(int n) {

}
}
```

### C Solution:

```
/*
* Problem: Closest Fair Integer
* 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
```

```
*/

int closestFair(int n) {

}
```

### Go Solution:

```
// Problem: Closest Fair Integer
// 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 closestFair(n int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Closest Fair Integer
// 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_fair(n: i32) -> i32 {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (closest-fair n)
(-> exact-integer? exact-integer?)
)
```