Problem 650: 2 Keys Keyboard

Problem Information

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

Problem Description

There is only one character

'A'

on the screen of a notepad. You can perform one of two operations on this notepad for each step:

Copy All: You can copy all the characters present on the screen (a partial copy is not allowed).

Paste: You can paste the characters which are copied last time.

Given an integer

n

, return

the minimum number of operations to get the character

'A'

exactly

n

times on the screen

.

Example 1:

Input:

n = 3

Output:

3

Explanation:

Initially, we have one character 'A'. In step 1, we use Copy All operation. In step 2, we use Paste operation to get 'AA'. In step 3, we use Paste operation to get 'AAA'.

Example 2:

Input:

n = 1

Output:

0

Constraints:

1 <= n <= 1000

Code Snippets

C++:

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



}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minSteps(int n) {

}
```

Go:

```
func minSteps(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (min-steps n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minSteps(int n) {

}
};
```

Java Solution:

```
/**
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minSteps(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: 2 Keys Keyboard
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minSteps(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minSteps(n: number): number {

};
```

C# Solution:

```
/*
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinSteps(int n) {

}
}
```

C Solution:

```
/*
* Problem: 2 Keys Keyboard
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minSteps(int n) {

}
```

Go Solution:

```
// Problem: 2 Keys Keyboard
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func minSteps(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: 2 Keys Keyboard
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-steps n)
(-> exact-integer? exact-integer?)
)
```