Problem 651: 4 Keys Keyboard

Problem Information

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

Problem Description

Imagine you have a special keyboard with the following keys:

A: Print one

'A'

on the screen.

Ctrl-A: Select the whole screen.

Ctrl-C: Copy selection to buffer.

Ctrl-V: Print buffer on screen appending it after what has already been printed.

Given an integer n, return

the maximum number of

'A'

you can print on the screen with

at most

n

presses on the keys

.

Example 1:

Input:

n = 3

Output:

3

Explanation:

We can at most get 3 A's on screen by pressing the following key sequence: A, A, A

Example 2:

Input:

n = 7

Output:

9

Explanation:

We can at most get 9 A's on screen by pressing following key sequence: A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V

Constraints:

1 <= n <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

}

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxA(int n) {

}
```

Go:

```
func maxA(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-a n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 4 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 maxA(int n) {

}
};
```

Java Solution:

```
/**
* Problem: 4 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 maxA(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: 4 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 maxA(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: 4 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 maxA = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: 4 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 maxA(n: number): number {

};
```

C# Solution:

```
/*
* Problem: 4 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 MaxA(int n) {

}
}
```

C Solution:

```
/*
* Problem: 4 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 maxA(int n) {

}
```

Go Solution:

```
// Problem: 4 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 maxA(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: 4 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 max_a(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-a n)
(-> exact-integer? exact-integer?)
)
```