Problem 3536: Maximum Product of Two Digits

Problem Information

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

Problem Description

You are given a positive integer

n

.

Return the

maximum

product of any two digits in

n

.

Note:

You may use the

same

digit twice if it appears more than once in

n

.

Example 1:

Input:

n = 31

Output:

3

Explanation:

The digits of

n

are

[3, 1]

.

The possible products of any two digits are:

3 * 1 = 3

.

The maximum product is 3.

Example 2:

Input:

n = 22

Output:

4

Explanation:

The digits of

n

are

[2, 2]

.

The possible products of any two digits are:

2 * 2 = 4

.

The maximum product is 4.

Example 3:

Input:

n = 124

Output:

8

Explanation:

The digits of

n

are

[1, 2, 4]

.

The possible products of any two digits are:

1 * 2 = 2

,

1 * 4 = 4

,

2 * 4 = 8

.

The maximum product is 8.

Constraints:

10 <= n <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxProduct(int n) {
}
}
```

C:

```
int maxProduct(int n) {

}
```

Go:

```
func maxProduct(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 maxProduct(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 maxProduct(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of Two Digits
Difficulty: Easy
Tags: math, sort

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 maxProduct(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 maxProduct = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 maxProduct(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 MaxProduct(int n) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product of Two Digits
* Difficulty: Easy
* Tags: math, sort
*
* 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 maxProduct(int n) {

}
```

Go Solution:

```
// Problem: Maximum Product of Two Digits
// Difficulty: Easy
// Tags: math, sort
//
// 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 maxProduct(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of Two Digits
// Difficulty: Easy
// Tags: math, sort
//
// 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 max_product(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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