Problem 3270: Find the Key of the Numbers

Problem Information

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

Problem Description

You are given three

positive

integers

num1

,

num2

, and

num3

.

The

key

of

num1

,

num2

, and

num3

is defined as a four-digit number such that:

Initially, if any number has

less than

four digits, it is padded with

leading zeros

.

The

i

th

digit (

1 <= i <= 4

) of the

key

is generated by taking the

smallest

digit among the

i

th

digits of

num1

,

num2

, and

num3

.

Return the

key

of the three numbers

without

leading zeros (

if any

).

Example 1:

Input:

num1 = 1, num2 = 10, num3 = 1000

Output:

0

Explanation:

On padding,

num1

becomes

"0001"

,

num2

becomes

"0010"

, and

num3

remains

"1000"

.

The

1

st

digit of the

key

is

min(0, 0, 1)

.

The

2

nd

digit of the

key

is

min(0, 0, 0)

.

The

3

rd

digit of the

key

is

min(0, 1, 0)

.

The

4

th

digit of the

key

is

min(1, 0, 0)

.

Hence, the

key

is

"0000"

, i.e. 0.

Example 2:

Input:

num1 = 987, num2 = 879, num3 = 798

Output:

777

Example 3:

Input:

num1 = 1, num2 = 2, num3 = 3

Output:

1

Constraints:

1 <= num1, num2, num3 <= 9999

Code Snippets

C++:

```
class Solution {
public:
int generateKey(int num1, int num2, int num3) {

}
};
```

Java:

```
class Solution {
public int generateKey(int num1, int num2, int num3) {

}
}
```

Python3:

```
class Solution:
def generateKey(self, num1: int, num2: int, num3: int) -> int:
```

Python:

```
class Solution(object):
def generateKey(self, num1, num2, num3):
```

```
"""
:type num1: int
:type num2: int
:type num3: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num1
* @param {number} num2
* @param {number} num3
* @return {number}
*/
var generateKey = function(num1, num2, num3) {

};
```

TypeScript:

```
function generateKey(num1: number, num2: number, num3: number): number {

};
```

C#:

```
public class Solution {
public int GenerateKey(int num1, int num2, int num3) {

}
}
```

C:

```
int generateKey(int num1, int num2, int num3) {

}
```

Go:

```
func generateKey(num1 int, num2 int, num3 int) int {

}
```

Kotlin:

```
class Solution {
fun generateKey(num1: Int, num2: Int, num3: Int): Int {

}
}
```

Swift:

```
class Solution {
func generateKey(_ num1: Int, _ num2: Int, _ num3: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn generate_key(num1: i32, num2: i32, num3: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num1
# @param {Integer} num2
# @param {Integer} num3
# @return {Integer}
def generate_key(num1, num2, num3)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer $num1
* @param Integer $num2
* @param Integer $num3
* @return Integer
*/
function generateKey($num1, $num2, $num3) {

}
}
```

Dart:

```
class Solution {
int generateKey(int num1, int num2, int num3) {

}
}
```

Scala:

```
object Solution {
def generateKey(num1: Int, num2: Int, num3: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_key(num1 :: integer, num2 :: integer, num3 :: integer) ::
integer
def generate_key(num1, num2, num3) do

end
end
```

Erlang:

```
-spec generate_key(Num1 :: integer(), Num2 :: integer(), Num3 :: integer())
-> integer().
generate_key(Num1, Num2, Num3) ->
.
```

Racket:

```
(define/contract (generate-key num1 num2 num3)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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 generateKey(int num1, int num2, int num3) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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 generateKey(int num1, int num2, int num3) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Find the Key of the Numbers
Difficulty: Easy
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 generateKey(self, num1: int, num2: int, num3: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def generateKey(self, num1, num2, num3):
"""
:type num1: int
:type num2: int
:type num3: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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} num1
* @param {number} num2
* @param {number} num3
* @return {number}
*/
var generateKey = function(num1, num2, num3) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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 generateKey(num1: number, num2: number, num3: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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 GenerateKey(int num1, int num2, int num3) {

}
}
```

C Solution:

```
/*
* Problem: Find the Key of the Numbers
* Difficulty: Easy
* 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 generateKey(int num1, int num2, int num3) {

}
```

Go Solution:

```
// Problem: Find the Key of the Numbers
// Difficulty: Easy
// 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 generateKey(num1 int, num2 int, num3 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun generateKey(num1: Int, num2: Int, num3: Int): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func generateKey(_ num1: Int, _ num2: Int, _ num3: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Key of the Numbers
// Difficulty: Easy
// 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 generate_key(num1: i32, num2: i32, num3: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num1
# @param {Integer} num2
# @param {Integer} num3
# @return {Integer}
def generate_key(num1, num2, num3)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $num1
* @param Integer $num2
* @param Integer $num3
* @return Integer
*/
function generateKey($num1, $num2, $num3) {

}
}
```

Dart Solution:

```
class Solution {
int generateKey(int num1, int num2, int num3) {

}
}
```

Scala Solution:

```
object Solution {
def generateKey(num1: Int, num2: Int, num3: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_key(num1 :: integer, num2 :: integer, num3 :: integer) ::
integer
def generate_key(num1, num2, num3) do

end
end
```

Erlang Solution:

```
-spec generate_key(Num1 :: integer(), Num2 :: integer(), Num3 :: integer())
-> integer().
```

```
generate_key(Num1, Num2, Num3) ->
.
```

Racket Solution:

```
(define/contract (generate-key num1 num2 num3)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```