Problem 1215: Stepping Numbers

Problem Information

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

Problem Description

A

stepping number

is an integer such that all of its adjacent digits have an absolute difference of exactly

1

.

For example,

321

is a

stepping number

while

421

is not.

Given two integers

low

and

high

, return

a sorted list of all the

stepping numbers

in the inclusive range

[low, high]

.

Example 1:

Input:

low = 0, high = 21

Output:

[0,1,2,3,4,5,6,7,8,9,10,12,21]

Example 2:

Input:

low = 10, high = 15

Output:

[10,12]

Constraints:

0 <= low <= high <= 2 * 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> countSteppingNumbers(int low, int high) {

}
};
```

Java:

```
class Solution {
public List<Integer> countSteppingNumbers(int low, int high) {

}
}
```

Python3:

```
class Solution:
def countSteppingNumbers(self, low: int, high: int) -> List[int]:
```

Python:

```
class Solution(object):
def countSteppingNumbers(self, low, high):
"""
:type low: int
:type high: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} low
* @param {number} high
* @return {number[]}
*/
var countSteppingNumbers = function(low, high) {

};
```

TypeScript:

```
function countSteppingNumbers(low: number, high: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> CountSteppingNumbers(int low, int high) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* countSteppingNumbers(int low, int high, int* returnSize) {

}
```

Go:

```
func countSteppingNumbers(low int, high int) []int {

}
```

Kotlin:

```
class Solution {
fun countSteppingNumbers(low: Int, high: Int): List<Int> {

}
}
```

Swift:

```swift
class Solution {
func countSteppingNumbers(_ low: Int, _ high: Int) -> [Int] {

}
}
```

Rust:

```rust
impl Solution {
pub fn count_stepping_numbers(low: i32, high: i32) -> Vec<i32> {

}
}
```

Ruby:

```ruby
# @param {Integer} low
# @param {Integer} high
# @return {Integer[]}
def count_stepping_numbers(low, high)

end
```

PHP:

```php
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @return Integer[]
*/
function countSteppingNumbers($low, $high) {

}
}
```

Dart:

```
class Solution {
List<int> countSteppingNumbers(int low, int high) {

}
}
```

Scala:

```
object Solution {
def countSteppingNumbers(low: Int, high: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_stepping_numbers(low :: integer, high :: integer) :: [integer]
def count_stepping_numbers(low, high) do

end
end
```

Erlang:

```
-spec count_stepping_numbers(Low :: integer(), High :: integer()) ->
[integer()].
count_stepping_numbers(Low, High) ->
.
```

Racket:

```
(define/contract (count-stepping-numbers low high)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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:
vector<int> countSteppingNumbers(int low, int high) {

}
};
```

Java Solution:

```
/**
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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 List<Integer> countSteppingNumbers(int low, int high) {

}
}
```

Python3 Solution:

```
"""
Problem: Stepping Numbers
Difficulty: Medium
Tags: math, sort, search
```

```
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 countSteppingNumbers(self, low: int, high: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countSteppingNumbers(self, low, high):
"""
:type low: int
:type high: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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} low
* @param {number} high
* @return {number[]}
*/
var countSteppingNumbers = function(low, high) {

};
```

TypeScript Solution:

```
/**
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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 countSteppingNumbers(low: number, high: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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 IList<int> CountSteppingNumbers(int low, int high) {

}
}
```

C Solution:

```
/*
* Problem: Stepping Numbers
* Difficulty: Medium
* Tags: math, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* countSteppingNumbers(int low, int high, int* returnSize) {

}
```

Go Solution:

```
// Problem: Stepping Numbers
// Difficulty: Medium
// Tags: math, sort, search
//
// 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 countSteppingNumbers(low int, high int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun countSteppingNumbers(low: Int, high: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func countSteppingNumbers(_ low: Int, _ high: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Stepping Numbers
// Difficulty: Medium
// Tags: math, sort, search
//
// 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 count_stepping_numbers(low: i32, high: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} low
# @param {Integer} high
# @return {Integer[]}
def count_stepping_numbers(low, high)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @return Integer[]
*/
function countSteppingNumbers($low, $high) {

}
}
```

Dart Solution:

```
class Solution {
List<int> countSteppingNumbers(int low, int high) {

}
}
```

Scala Solution:

```
object Solution {
def countSteppingNumbers(low: Int, high: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_stepping_numbers(low :: integer, high :: integer) :: [integer]
def count_stepping_numbers(low, high) do

end
end
```

Erlang Solution:

```
-spec count_stepping_numbers(Low :: integer(), High :: integer()) ->
[integer()].
count_stepping_numbers(Low, High) ->
.
```

Racket Solution:

```
(define/contract (count-stepping-numbers low high)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```