Problem 247: Strobogrammatic Number II

Problem Information

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

Problem Description

Given an integer

n

, return all the

strobogrammatic numbers

that are of length

n

. You may return the answer in

any order

.

A

strobogrammatic number

is a number that looks the same when rotated

180

degrees (looked at upside down).

Example 1:

Input:

n = 2

Output:

["11","69","88","96"]

Example 2:

Input:

n = 1

Output:

["0","1","8"]

Constraints:

1 <= n <= 14

Code Snippets

C++:

```
class Solution {
public:
vector<string> findStrobogrammatic(int n) {

}
};
```

Java:

```
class Solution {
public List<String> findStrobogrammatic(int n) {

}
}
```

Python3:

```
class Solution:
def findStrobogrammatic(self, n: int) -> List[str]:
```

Python:

```
class Solution(object):
def findStrobogrammatic(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {string[]}
*/
var findStrobogrammatic = function(n) {

};
```

TypeScript:

```
function findStrobogrammatic(n: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FindStrobogrammatic(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findStrobogrammatic(int n, int* returnSize) {

}
```

Go:

```
func findStrobogrammatic(n int) []string {

}
```

Kotlin:

```
class Solution {
fun findStrobogrammatic(n: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func findStrobogrammatic(_ n: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_strobogrammatic(n: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String[]}
```

```
def find_strobogrammatic(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function findStrobogrammatic($n) {

}
}
```

Dart:

```
class Solution {
List<String> findStrobogrammatic(int n) {

}
}
```

Scala:

```
object Solution {
def findStrobogrammatic(n: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_strobogrammatic(n :: integer) :: [String.t]
def find_strobogrammatic(n) do

end
end
```

Erlang:

```
-spec find_strobogrammatic(N :: integer()) -> [unicode:unicode_binary()].
find_strobogrammatic(N) ->
.
```

Racket:

```
(define/contract (find-strobogrammatic n)
(-> exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> findStrobogrammatic(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> findStrobogrammatic(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Strobogrammatic Number II
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findStrobogrammatic(self, n: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findStrobogrammatic(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {string[]}
*/
var findStrobogrammatic = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findStrobogrammatic(n: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public IList<string> FindStrobogrammatic(int n) {

}
}
```

C Solution:

```
/*
* Problem: Strobogrammatic Number II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findStrobogrammatic(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Strobogrammatic Number II
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findStrobogrammatic(n int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findStrobogrammatic(n: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func findStrobogrammatic(_ n: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Strobogrammatic Number II
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_strobogrammatic(n: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String[]}
def find_strobogrammatic(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function findStrobogrammatic($n) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findStrobogrammatic(int n) {

}
}
```

Scala Solution:

```
object Solution {
def findStrobogrammatic(n: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_strobogrammatic(n :: integer) :: [String.t]
def find_strobogrammatic(n) do

end
end
```

Erlang Solution:

```
-spec find_strobogrammatic(N :: integer()) -> [unicode:unicode_binary()].
find_strobogrammatic(N) ->
.
```

Racket Solution:

```
(define/contract (find-strobogrammatic n)
(-> exact-integer? (listof string?))
)
```