Problem 2094: Finding 3-Digit Even Numbers

Problem Information

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

Problem Description

You are given an integer array

digits

, where each element is a digit. The array may contain duplicates.

You need to find

all

the

unique

integers that follow the given requirements:

The integer consists of the

concatenation

of

three

elements from

digits

in

any

arbitrary order.

The integer does not have

leading zeros

.

The integer is

even

.

For example, if the given

digits

were

[1, 2, 3]

, integers

132

and

312

follow the requirements.

Return

a

sorted

array of the unique integers.

Example 1:

Input:

digits = [2,1,3,0]

Output:

[102,120,130,132,210,230,302,310,312,320]

Explanation:

All the possible integers that follow the requirements are in the output array. Notice that there are no

odd

integers or integers with

leading zeros

.

Example 2:

Input:

digits = [2,2,8,8,2]

Output:

[222,228,282,288,822,828,882]

Explanation:

The same digit can be used as many times as it appears in digits. In this example, the digit 8 is used twice each time in 288, 828, and 882.

Example 3:

Input:

digits = [3,7,5]

Output:

[]

Explanation:

No

even

integers can be formed using the given digits.

Constraints:

3 <= digits.length <= 100

0 <= digits[i] <= 9

Code Snippets

C++:

```
class Solution {
public:
vector<int> findEvenNumbers(vector<int>& digits) {

}
};
```

Java:

```
class Solution {
public int[] findEvenNumbers(int[] digits) {

}
}
```

Python3:

```
class Solution:
def findEvenNumbers(self, digits: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def findEvenNumbers(self, digits):
"""
:type digits: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} digits
* @return {number[]}
*/
var findEvenNumbers = function(digits) {

};
```

TypeScript:

```
function findEvenNumbers(digits: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] FindEvenNumbers(int[] digits) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findEvenNumbers(int* digits, int digitsSize, int* returnSize) {

}
```

Go:

```
func findEvenNumbers(digits []int) []int {

}
```

Kotlin:

```
class Solution {
fun findEvenNumbers(digits: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func findEvenNumbers(_ digits: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_even_numbers(digits: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} digits
# @return {Integer[]}
def find_even_numbers(digits)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $digits
* @return Integer[]
*/
function findEvenNumbers($digits) {

}
}
```

Dart:

```
class Solution {
List<int> findEvenNumbers(List<int> digits) {

}
}
```

Scala:

```
object Solution {
def findEvenNumbers(digits: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_even_numbers(digits :: [integer]) :: [integer]
def find_even_numbers(digits) do

end
end
```

Erlang:

```
-spec find_even_numbers(Digits :: [integer()]) -> [integer()].
find_even_numbers(Digits) ->
.
```

Racket:

```
(define/contract (find-even-numbers digits)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> findEvenNumbers(vector<int>& digits) {

}
};
```

Java Solution:

```
/**
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] findEvenNumbers(int[] digits) {

}
}
```

Python3 Solution:

```
"""
Problem: Finding 3-Digit Even Numbers
Difficulty: Easy
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def findEvenNumbers(self, digits):
"""
:type digits: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} digits
* @return {number[]}
*/
var findEvenNumbers = function(digits) {

};
```

TypeScript Solution:

```
/**
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findEvenNumbers(digits: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] FindEvenNumbers(int[] digits) {

}
}
```

C Solution:

```
/*
* Problem: Finding 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findEvenNumbers(int* digits, int digitsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Finding 3-Digit Even Numbers
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findEvenNumbers(digits []int) []int {
}
```

Kotlin Solution:

```
class Solution {
fun findEvenNumbers(digits: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func findEvenNumbers(_ digits: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Finding 3-Digit Even Numbers
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_even_numbers(digits: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} digits
# @return {Integer[]}
def find_even_numbers(digits)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $digits
* @return Integer[]
*/
function findEvenNumbers($digits) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findEvenNumbers(List<int> digits) {

}
}
```

Scala Solution:

```
object Solution {
def findEvenNumbers(digits: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_even_numbers(digits :: [integer]) :: [integer]
def find_even_numbers(digits) do

end
end
```

Erlang Solution:

```
-spec find_even_numbers(Digits :: [integer()]) -> [integer()].
find_even_numbers(Digits) ->
.
```

Racket Solution:

```
(define/contract (find-even-numbers digits)
(-> (listof exact-integer?) (listof exact-integer?))
)
```