Problem 401: Binary Watch

Problem Information

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

Problem Description

A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

For example, the below binary watch reads

"4:51"

.

Given an integer

turnedOn

which represents the number of LEDs that are currently on (ignoring the PM), return

all possible times the watch could represent

. You may return the answer in

any order

.

The hour must not contain a leading zero.

For example,

"01:00"

is not valid. It should be

"1:00"

.

The minute must consist of two digits and may contain a leading zero.

For example,

"10:2"

is not valid. It should be

"10:02"

.

Example 1:

Input:

turnedOn = 1

Output:

["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]

Example 2:

Input:

turnedOn = 9

Output:

[]

Constraints:

0 <= turnedOn <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<string> readBinaryWatch(int turnedOn) {

}
};
```

Java:

```
class Solution {
public List<String> readBinaryWatch(int turnedOn) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function readBinaryWatch(turnedOn: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> ReadBinaryWatch(int turnedOn) {

}
}
```

C:

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

}
```

Go:

```
func readBinaryWatch(turnedOn int) []string {

}
```

Kotlin:

```
class Solution {
fun readBinaryWatch(turnedOn: Int): List<String> {

}
```

```
}
```

Swift:

```
class Solution {
func readBinaryWatch(_ turnedOn: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn read_binary_watch(turned_on: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Integer} turned_on
# @return {String[]}
def read_binary_watch(turned_on)

end
```

PHP:

```
class Solution {

/**
* @param Integer $turnedOn
* @return String[]
*/
function readBinaryWatch($turnedOn) {

}
}
```

Dart:

```
class Solution {
List<String> readBinaryWatch(int turnedOn) {

}
}
```

Scala:

```
object Solution {
def readBinaryWatch(turnedOn: Int): List[String] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec read_binary_watch(TurnedOn :: integer()) -> [unicode:unicode_binary()].
read_binary_watch(TurnedOn) ->
.
```

Racket:

```
(define/contract (read-binary-watch turnedOn)
(-> exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Watch
```

```
* Difficulty: Easy
* Tags: general
*
* 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<string> readBinaryWatch(int turnedOn) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Watch
* Difficulty: Easy
* Tags: general
*
* 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<String> readBinaryWatch(int turnedOn) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Watch
Difficulty: Easy
Tags: general

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 readBinaryWatch(self, turnedOn: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Binary Watch
* Difficulty: Easy
* Tags: general
*
* 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} turnedOn
* @return {string[]}
*/
var readBinaryWatch = function(turnedOn) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Watch
* Difficulty: Easy
* Tags: general
*
* 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 readBinaryWatch(turnedOn: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Binary Watch
* Difficulty: Easy
* Tags: general
*
* 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<string> ReadBinaryWatch(int turnedOn) {

}
}
```

C Solution:

```
/*
* Problem: Binary Watch
* Difficulty: Easy
* Tags: general
*
* 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().
*/
char** readBinaryWatch(int turnedOn, int* returnSize) {

}
```

Go Solution:

```
// Problem: Binary Watch
// Difficulty: Easy
// Tags: general
//
// 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 readBinaryWatch(turnedOn int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun readBinaryWatch(turnedOn: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func readBinaryWatch(_ turnedOn: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Binary Watch
// Difficulty: Easy
// Tags: general
//
// 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 read_binary_watch(turned_on: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Integer} turned_on
# @return {String[]}
def read_binary_watch(turned_on)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $turnedOn
* @return String[]
*/
function readBinaryWatch($turnedOn) {

}
}
```

Dart Solution:

```
class Solution {
List<String> readBinaryWatch(int turnedOn) {

}
}
```

Scala Solution:

```
object Solution {
def readBinaryWatch(turnedOn: Int): List[String] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec read_binary_watch(TurnedOn :: integer()) -> [unicode:unicode_binary()].
read_binary_watch(TurnedOn) ->
.
```

Racket Solution:

```
(define/contract (read-binary-watch turnedOn)
(-> exact-integer? (listof string?))
)
```