

Problem 89: Gray Code

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

An

n -bit gray code sequence

is a sequence of

2^n

integers where:

Every integer is in the

range $[0, 2^n - 1]$

inclusive

range

$[0, 2^n - 1]$

such that

the

sequence

The first integer is

0

,

An integer appears

no more than once

in the sequence,

The binary representation of every pair of

adjacent

integers differs by

exactly one bit

, and

The binary representation of the

first

and

last

integers differs by

exactly one bit

.

Given an integer

n

, return

any valid

n-bit gray code sequence

.

Example 1:

Input:

n = 2

Output:

[0,1,3,2]

Explanation:

The binary representation of [0,1,3,2] is [00,01,11,10]. - 0

0

and 0

1

differ by one bit -

0

1 and

1

1 differ by one bit - 1

1

and 1

0

differ by one bit -

1

0 and

0

0 differ by one bit [0,2,3,1] is also a valid gray code sequence, whose binary representation is [00,10,11,01]. -

0

0 and

1

0 differ by one bit - 1

0

and 1

1

differ by one bit -

1

1 and

0

1 differ by one bit - 0

1

and 0

0

differ by one bit

Example 2:

Input:

n = 1

Output:

[0,1]

Constraints:

$1 \leq n \leq 16$

Code Snippets

C++:

```
class Solution {  
public:  
    vector<int> grayCode(int n) {  
  
    }  
};
```

Java:

```
class Solution {  
    public List<Integer> grayCode(int n) {
```

```
}  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function grayCode(n: number): number[] {  
  
};
```

C#:

```
public class Solution {  
    public IList<int> GrayCode(int n) {  
  
    }  
}
```

C:

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

}
```

Go:

```
func grayCode(n int) []int {

}
```

Kotlin:

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

    }
}
```

Swift:

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

    }
}
```

Rust:

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

    }
}
```

Ruby:

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

```
def gray_code(n)

end
```

PHP:

```
class Solution {

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

    }

}
```

Dart:

```
class Solution {
  List<int> grayCode(int n) {

  }
}
```

Scala:

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

  }
}
```

Elixir:

```
defmodule Solution do
  @spec gray_code(n :: integer) :: [integer]
  def gray_code(n) do

  end
end
```

Erlang:

```
-spec gray_code(N :: integer()) -> [integer()].  
gray_code(N) ->  
. 
```

Racket:

```
(define/contract (gray-code n)  
  (-> exact-integer? (listof exact-integer?))  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Gray Code  
 * Difficulty: Medium  
 * 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:  
    vector<int> grayCode(int n) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Gray Code  
 * Difficulty: Medium  
 * 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 List<Integer> grayCode(int n) {

}

}

```

Python3 Solution:

```

"""
Problem: Gray Code
Difficulty: Medium
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 grayCode(self, n: int) -> List[int]:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Gray Code
 * Difficulty: Medium

```

```

* 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} n
* @return {number[]}
*/
var grayCode = function(n) {

};

```

TypeScript Solution:

```

/**
* Problem: Gray Code
* Difficulty: Medium
* 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 grayCode(n: number): number[] {

};

```

C# Solution:

```

/*
* Problem: Gray Code
* Difficulty: Medium
* 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 IList<int> GrayCode(int n) {

    }
}

```

C Solution:

```

/*
 * Problem: Gray Code
 * Difficulty: Medium
 * 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
 */

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

}

```

Go Solution:

```

// Problem: Gray Code
// Difficulty: Medium
// 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 grayCode(n int) []int {

}

```

Kotlin Solution:

```
class Solution {  
    fun grayCode(n: Int): List<Int> {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func grayCode(_ n: Int) -> [Int] {  
  
    }  
}
```

Rust Solution:

```
// Problem: Gray Code  
// Difficulty: Medium  
// 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 gray_code(n: i32) -> Vec<i32> {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer} n  
# @return {Integer[]}  
def gray_code(n)  
  
end
```

PHP Solution:

```

class Solution {

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

    }

}

```

Dart Solution:

```

class Solution {
  List<int> grayCode(int n) {

  }

}

```

Scala Solution:

```

object Solution {
  def grayCode(n: Int): List[Int] = {

  }

}

```

Elixir Solution:

```

defmodule Solution do
  @spec gray_code(n :: integer) :: [integer]
  def gray_code(n) do

  end

end

```

Erlang Solution:

```

-spec gray_code(N :: integer()) -> [integer()].
gray_code(N) ->

.

```

Racket Solution:

```
(define/contract (gray-code n)
  (-> exact-integer? (listof exact-integer?))
)
```