Problem 1652: Defuse the Bomb

Problem Information

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

Problem Description

You have a bomb to defuse, and your time is running out! Your informer will provide you with a

circular

array

code

of length of

n

and a key

k

.

To decrypt the code, you must replace every number. All the numbers are replaced

simultaneously

.

If

k > 0

, replace the

i

th

number with the sum of the

next

k

numbers.

If

k < 0

, replace the

i

th

number with the sum of the

previous

k

numbers.

If

k == 0

, replace the

i

th

number with

0

.

As

code

is circular, the next element of

code[n-1]

is

code[0]

, and the previous element of

code[0]

is

code[n-1]

.

Given the

circular

array

code

and an integer key

k

, return

the decrypted code to defuse the bomb

!

Example 1:

Input:

code = [5,7,1,4], k = 3

Output:

[12,10,16,13]

Explanation:

Each number is replaced by the sum of the next 3 numbers. The decrypted code is [7+1+4, 1+4+5, 4+5+7, 5+7+1]. Notice that the numbers wrap around.

Example 2:

Input:

code = [1,2,3,4], k = 0

Output:

[0,0,0,0]

Explanation:

When k is zero, the numbers are replaced by 0.

Example 3:

Input:

code = [2,4,9,3], k = -2

Output:

[12,5,6,13]

Explanation:

The decrypted code is [3+9, 2+3, 4+2, 9+4]. Notice that the numbers wrap around again. If k is negative, the sum is of the

previous

numbers.

Constraints:

n == code.length

1 <= n <= 100

1 <= code[i] <= 100

-(n - 1) <= k <= n - 1

Code Snippets

C++:

```
class Solution {
public:
vector<int> decrypt(vector<int>& code, int k) {
}
};
```

Java:

```
class Solution {
public int[] decrypt(int[] code, int k) {

}
}
```

Python3:

```
class Solution:
def decrypt(self, code: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def decrypt(self, code, k):
"""
:type code: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} code
* @param {number} k
* @return {number[]}
*/
var decrypt = function(code, k) {

};
```

TypeScript:

```
function decrypt(code: number[], k: number): number[] {
};
```

C#:

```
public class Solution {
public int[] Decrypt(int[] code, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* decrypt(int* code, int codeSize, int k, int* returnSize) {

}
```

Go:

```
func decrypt(code []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun decrypt(code: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func decrypt(_ code: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn decrypt(code: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} code
# @param {Integer} k
# @return {Integer[]}
def decrypt(code, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $code
* @param Integer $k
* @return Integer[]
*/
function decrypt($code, $k) {

}
}
```

Dart:

```
class Solution {
List<int> decrypt(List<int> code, int k) {

}
}
```

Scala:

```
object Solution {
def decrypt(code: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec decrypt(code :: [integer], k :: integer) :: [integer]
def decrypt(code, k) do

end
end
```

Erlang:

```
-spec decrypt(Code :: [integer()], K :: integer()) -> [integer()].
decrypt(Code, K) ->
.
```

Racket:

```
(define/contract (decrypt code k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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<int> decrypt(vector<int>& code, int k) {
}
};
```

Java Solution:

```
/**
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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 int[] decrypt(int[] code, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Defuse the Bomb
Difficulty: Easy
Tags: array

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 decrypt(self, code: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def decrypt(self, code, k):
"""
:type code: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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[]} code
* @param {number} k
* @return {number[]}
*/
var decrypt = function(code, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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 decrypt(code: number[], k: number): number[] {
};
```

C# Solution:

```
/*
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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 int[] Decrypt(int[] code, int k) {

}
}
```

C Solution:

```
/*
* Problem: Defuse the Bomb
* Difficulty: Easy
* Tags: array
*
* 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().
*/
int* decrypt(int* code, int codeSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Defuse the Bomb
// Difficulty: Easy
// Tags: array
//
// 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 decrypt(code []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun decrypt(code: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func decrypt(_ code: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Defuse the Bomb
// Difficulty: Easy
// Tags: array
//
// 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 decrypt(code: Vec<i32>, k: i32) -> Vec<i32> {

}

}
```

Ruby Solution:

```
# @param {Integer[]} code
# @param {Integer} k
# @return {Integer[]}
def decrypt(code, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $code
* @param Integer $k
* @return Integer[]
*/
function decrypt($code, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> decrypt(List<int> code, int k) {

}
}
```

Scala Solution:

```
object Solution {
def decrypt(code: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec decrypt(code :: [integer], k :: integer) :: [integer]
def decrypt(code, k) do

end
end
```

Erlang Solution:

```
-spec decrypt(Code :: [integer()], K :: integer()) -> [integer()].
decrypt(Code, K) ->
.
```

Racket Solution:

```
(define/contract (decrypt code k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```