Problem 278: First Bad Version

Problem Information

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

Problem Description

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have

n

versions

[1, 2, ..., n]

and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API

bool isBadVersion(version)

which returns whether

version

is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

Input:

n = 5, bad = 4

Output:

4

Explanation:

call isBadVersion(3) -> false call isBadVersion(5) -> true call isBadVersion(4) -> true Then 4 is the first bad version.

Example 2:

Input:

n = 1, bad = 1

Output:

1

Constraints:

1 <= bad <= n <= 2

31

- 1

Code Snippets

C++:

```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);
```

```
class Solution {
public:
int firstBadVersion(int n) {

}
};
```

Java:

```
/* The isBadVersion API is defined in the parent class VersionControl.
boolean isBadVersion(int version); */

public class Solution extends VersionControl {
public int firstBadVersion(int n) {

}
}
```

Python3:

```
# The isBadVersion API is already defined for you.
# def isBadVersion(version: int) -> bool:

class Solution:
def firstBadVersion(self, n: int) -> int:
```

Python:

```
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution(object):
def firstBadVersion(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for isBadVersion()
*
* @param {integer} version number
* @return {boolean} whether the version is bad
* isBadVersion = function(version) {
* ...
* };
*/

/**
* @param {function} isBadVersion()
* @return {function}
*/
var solution = function(isBadVersion) {
/**
* @param {integer} n Total versions
* @return {integer} The first bad version
*/
return function(n) {

};
};
```

TypeScript:

```
/**
* The knows API is defined in the parent class Relation.
* isBadVersion(version: number): boolean {
* ...
* };
*/

var solution = function(isBadVersion: any) {

return function(n: number): number {

};
};
```

C#:

```
/* The isBadVersion API is defined in the parent class VersionControl.
bool IsBadVersion(int version); */

public class Solution : VersionControl {
public int FirstBadVersion(int n) {

}
}
```

C:

```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

int firstBadVersion(int n) {

}
```

Go:

```
/**
* Forward declaration of isBadVersion API.
* @param version your guess about first bad version
* @return true if current version is bad
* false if current version is good
* func isBadVersion(version int) bool;
*/

func firstBadVersion(n int) int {

}
```

Kotlin:

```
/* The isBadVersion API is defined in the parent class VersionControl.
fun isBadVersion(version: Int) : Boolean {} */

class Solution: VersionControl() {
override fun firstBadVersion(n: Int) : Int {

}
}
```

Swift:

```
/**
* The knows API is defined in the parent class VersionControl.
* func isBadVersion(_ version: Int) -> Bool{}
*/

class Solution : VersionControl {
func firstBadVersion(_ n: Int) -> Int {

}
}
```

Rust:

```
// The API isBadVersion is defined for you.
// isBadVersion(version:i32)-> bool;
// to call it use self.isBadVersion(version)

impl Solution {
pub fn first_bad_version(&self, n: i32) -> i32 {

}
}
```

Ruby:

```
# The is_bad_version API is already defined for you.
# @param {Integer} version
# @return {boolean} whether the version is bad
# def is_bad_version(version):

# @param {Integer} n
# @return {Integer}
def first_bad_version(n)

end
```

PHP:

```
/* The isBadVersion API is defined in the parent class VersionControl.
public function isBadVersion($version){} */
```

```
class Solution extends VersionControl {
/**
* @param Integer $n
* @return Integer
*/
function firstBadVersion($n) {

}
}
```

Scala:

```
/* The isBadVersion API is defined in the parent class VersionControl.
def isBadVersion(version: Int): Boolean = {} */

class Solution extends VersionControl {
def firstBadVersion(n: Int): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
```

```
int firstBadVersion(int n) {

}
};
```

Java Solution:

```
/**
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

/* The isBadVersion API is defined in the parent class VersionControl.
boolean isBadVersion(int version); */

public class Solution extends VersionControl {
public int firstBadVersion(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: First Bad Version
Difficulty: Easy
Tags: search

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
"""

# The isBadVersion API is already defined for you.
# def isBadVersion(version: int) -> bool:
```

```
class Solution:
def firstBadVersion(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution(object):
def firstBadVersion(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

/**
* Definition for isBadVersion()
*
* @param {integer} version number
* @return {boolean} whether the version is bad
* isBadVersion = function(version) {
* ...
* };
*/
```

```
/**
* @param {function} isBadVersion()
* @return {function}
*/
var solution = function(isBadVersion) {
/**
* @param {integer} n Total versions
* @return {integer} The first bad version
*/
return function(n) {

};
};
```

TypeScript Solution:

```
/**
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

/**
* The knows API is defined in the parent class Relation.
* isBadVersion(version: number): boolean {
* ...
* };
*/

var solution = function(isBadVersion: any) {

return function(n: number): number {

};
};
```

C# Solution:

```
/*
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

/* The isBadVersion API is defined in the parent class VersionControl.
bool IsBadVersion(int version); */

public class Solution : VersionControl {
public int FirstBadVersion(int n) {

}
}
```

C Solution:

```
/*
* Problem: First Bad Version
* Difficulty: Easy
* Tags: search
*
* 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
*/

// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

int firstBadVersion(int n) {

}
```

Go Solution:

```
// Problem: First Bad Version
// Difficulty: Easy
// Tags: search
//
// 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

/**
* Forward declaration of isBadVersion API.
* @param version your guess about first bad version
* @return true if current version is bad
* false if current version is good
* func isBadVersion(version int) bool;
*/

func firstBadVersion(n int) int {

}
```

Kotlin Solution:

```
/* The isBadVersion API is defined in the parent class VersionControl.
fun isBadVersion(version: Int) : Boolean {} */

class Solution: VersionControl() {
override fun firstBadVersion(n: Int) : Int {

}
}
```

Swift Solution:

```
/**
* The knows API is defined in the parent class VersionControl.
* func isBadVersion(_ version: Int) -> Bool{}
*/

class Solution : VersionControl {
func firstBadVersion(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: First Bad Version
// Difficulty: Easy
// Tags: search
//
// 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

// The API isBadVersion is defined for you.
// isBadVersion(version:i32)-> bool;
// to call it use self.isBadVersion(version)

impl Solution {
pub fn first_bad_version(&self, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# The is_bad_version API is already defined for you.
# @param {Integer} version
# @return {boolean} whether the version is bad
# def is_bad_version(version):

# @param {Integer} n
# @return {Integer}
def first_bad_version(n)

end
```

PHP Solution:

```
/* The isBadVersion API is defined in the parent class VersionControl.
public function isBadVersion($version){} */

class Solution extends VersionControl {
/**
* @param Integer $n
* @return Integer
*/
function firstBadVersion($n) {

}
}
```

Scala Solution:

```
/* The isBadVersion API is defined in the parent class VersionControl.
def isBadVersion(version: Int): Boolean = {} */

class Solution extends VersionControl {
def firstBadVersion(n: Int): Int = {

}
}
```