<https://leetcode.com/problems/first-bad-version/>

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

**Attempt 1: 2022-10-01**

**Solution 1: 10min, Binary Search Find Target First Occurrence (template based on while(lo <= hi), refer to L704.Binary Search)**

/\* The isBadVersion API is defined in the parent class VersionControl.

      boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int lo = 1;

        int hi = n;

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            if(isBadVersion(mid)) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        // No check required for non-existing answer return -1 case, because the problem

        // guaranteed there must be a 'T' exist in the input, not all 'F'

        //if(!isBadVersion(lo)) {

        //    return -1;

        //}

        return lo;

    }

}

Space Complexity: O(1)

Time Complexity: O(logn)

**Refer to**

<https://leetcode.com/problems/first-bad-version/discuss/606080/Java-Binary-Search-Clean-code-O(logN)>

public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int left = 1, right = n, ans = -1;

        while (left <= right) {

            int mid = left + (right - left) / 2; // to avoid overflow incase (left+right)>2147483647

            if (isBadVersion(mid)) {

                ans = mid; // record mid as current answer

                right = mid - 1; // try to find smaller version in the left side

            } else {

                left = mid + 1; // try to find in the right side

            }

        }

        return ans;

    }

}

**How while(lo <= hi) work with hi = mid - 1 and lo = mid + 1 to find first bad version ?**

Template:

while lo <= hi:

  mid = (lo+hi)/2

  if nums[mid] == target:

    return mid

  if nums[mid] > target:

    hi = mid-1

  else:

    lo = mid+1

return -1

=================================================================================

e.g

isBadVersion result -> FFTTTT

index start from 1  -> 123456

Find left most 'T'

while(lo <= hi) -> ending condition lo > hi, means no interval between ...hi|lo...

Round 1:

lo=1,hi=6 -> mid=3

isBadVersion(3)=T -> hi=mid-1=2 (hi=mid-1 means skip 'mid')

Round 2:

lo=1,hi=2 -> mid=1

isBadVersion(1)=F -> lo=mid+1=2 (lo=mid+1 means skip 'mid')

Round 3:

lo=2,hi=2 -> mid=2

isBadVersion(2)=F -> lo=mid+1=3 (lo=mid+1 means skip 'mid')

Now lo > hi while loop end, lo is the first position for isBadVersion return 'T',

no check required for non-existing answer return -1 case, because the problem guaranteed

there must be a 'T' exist in the input, not all 'F', return lo for first position

**Solution 2:  10min, Binary Search Find Target First Occurrence (template based on while(lo < hi)**

/\* The isBadVersion API is defined in the parent class VersionControl.

      boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int lo = 1;

        int hi = n;

        while(lo < hi) {

            int mid = lo + (hi - lo) / 2;

            if(isBadVersion(mid)) {

                hi = mid;

            } else {

                lo = mid + 1;

            }

        }

        return lo;

    }

}

Space Complexity: O(1)

Time Complexity: O(logn)

**How while(lo < hi) work with hi = mid and lo = mid + 1 to find first bad version ?**

**Refer to**

<https://leetcode.com/problems/first-bad-version/discuss/71386/An-clear-way-to-use-binary-search>

public int firstBadVersionLeft(int n) {

int i = 1;

int j = n;

while (i < j) {

int mid = i + (j - i) / 2;

if (isBadVersion(mid)) {

j = mid;

} else {

i = mid + 1;

}

}

return i;

}

**Step by Step**

Template:

while lo < hi:

  mid = (lo+hi)/2

  if function(mid):

    hi = mid

  else:

    lo = mid+1

return lo

=================================================================================

e.g

isBadVersion result -> FFTTTT

index start from 1  -> 123456

Find left most 'T'

while(lo < hi) -> ending condition lo == hi, means lo and hi overlap at same position ... hi ...

                                                                                          lo

Round 1:

lo=1,hi=6 -> mid=3

isBadVersion(3)=T -> hi=mid=3 (hi=mid means include 'mid')

Round 2:

lo=1,hi=3 -> mid=2

isBadVersion(2)=F -> lo=mid+1=3 (lo=mid+1 means skip 'mid')

Now lo == hi while loop end, and hi can equal to mid, which means include the final answer,

return either hi or lo is fine

**Conclusion**

<https://leetcode.com/problems/first-bad-version/discuss/71296/O(lgN)-simple-Java-solution/166412>

I think when termination condition is left < right, you need to set hi = mid - 1, lo = mid; when it's left <= right, you need to set lo = mid + 1 & hi = mid - 1

**Solution 3: Binary Search Find Target First Occurrence (template based on while(lo + 1 < hi)**

/\* The isBadVersion API is defined in the parent class VersionControl.

      boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int lo = 1;

        int hi = n;

        while(lo + 1 < hi) {

            int mid = lo + (hi - lo) / 2;

            if(isBadVersion(mid)) {

                hi = mid;

            } else {

                lo = mid;

            }

        }

        if(isBadVersion(lo)) {

            return lo;

        }

        return hi;

    }

}

Space Complexity: O(1)

Time Complexity: O(logn)

**Follow Up: Find Last Bad Version ?**

**Solution 1. while(lo <= hi) template**

/\* The isBadVersion API is defined in the parent class VersionControl.

      boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

    public int lastBadVersion(int n) {

        int lo = 1;

        int hi = n;

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            if(isBadVersion(mid)) {

                // Discard left half of 'mid' since we pursue last bad version position

                lo = mid + 1;

            } else {

                hi = mid - 1;

            }

        }

        // No check required for non-existing answer return -1 case, because the problem

        // guaranteed there must be a 'T' exist in the input, not all 'F'

        //if(!isBadVersion(lo)) {

        //    return -1;

        //}

        return hi;

    }

}

Space Complexity: O(1)

Time Complexity: O(logn)

**Refer to**

**L34.Find First and Last Position of Element in Sorted Array template 1**

  private int findEndingPos(int[] nums, int target) {

        int lo = 0;

        int hi = nums.length - 1;

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            if(nums[mid] > target) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        if(hi < 0 || nums[hi] != target) {

            return -1;

        }

        return hi;

    }

**How while(lo <= hi) work with hi = mid - 1 and lo = mid + 1 to find last bad version ?**

e.g

isBadVersion result -> FFTTTT

index start from 1  -> 123456

Find right most 'T'

while(lo <= hi) -> ending condition lo > hi, means no interval between ...hi|lo...

Round 1:

lo=1,hi=6 -> mid=3

isBadVersion(3)=T -> lo=mid+1=4 (lo=mid+1 means skip 'mid')

Round 2:

lo=4,hi=6 -> mid=5

isBadVersion(5)=T -> lo=mid+1=6 (lo=mid+1 means skip 'mid')

Round 3:

lo=6,hi=6 -> mid=6

isBadVersion(6)=T -> lo=mid+1=7 (lo=mid+1 means skip 'mid')

Now lo > hi while loop end, hi is the last position for isBadVersion return 'T',

no check required for non-existing answer return -1 case, because the problem guaranteed

there must be a 'T' exist in the input, not all 'F', return hi for last position

**Solution 2. while(lo < hi) template**

/\* The isBadVersion API is defined in the parent class VersionControl.

      boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int lo = 1;

        int hi = n;

        while(lo < hi) {

            int mid = lo + (hi - lo) / 2 + 1;

            if(isBadVersion(mid)) {

                lo = mid;

            } else {

                hi = mid - 1;

            }

        }

        return hi;

    }

}

Space Complexity: O(1)

Time Complexity: O(logn)

**How while(lo < hi) work with hi = mid  and lo = mid + 1 to find last good version ?**

**Refer to**

<https://leetcode.com/problems/first-bad-version/discuss/71386/An-clear-way-to-use-binary-search>

It is obvious that the version would looks like the following:

FFTTTT (The first two are correct version, the rest are bad ones)

To find the right most F we need to notice that since we are looking for the first bad version not the last correct version, we need to return the (position when the binary search stop) + 1

public int firstBadVersion(int n) {

if (isBadVersion(1)) {

return 1;

}

int i = 1;

int j = n;

while (i < j) {

int mid = i + (j - i) / 2 + 1;

if (!isBadVersion(mid)) {

i = mid;

} else {

j = mid - 1;

}

}

// To find the right most F we need to notice that since we are looking for the first bad version

// not the last correct version, we need to return the (position when the binary search stop) + 1

return j + 1;

}

**How while(lo < hi) work with hi = mid  and lo = mid + 1 to find last bad version ?**

**Refer to**

<https://leetcode.com/problems/first-bad-version/discuss/71386/An-clear-way-to-use-binary-search>

It is obvious that the version would looks like the following:

FFTTTT (The first two are correct version, the rest are bad ones)

To find the right most T we need to notice that since we are looking for the first bad version not the last correct version, we need to return the (position when the binary search stop) + 1

public int firstBadVersion(int n) {

if (isBadVersion(1)) {

return 1;

}

int i = 1;

int j = n;

while (i < j) {

int mid = i + (j - i) / 2 + 1;

if (!isBadVersion(mid)) {

j = mid - 1;

} else {

i = mid;

}

}

return i;

}

**Step by Step**

e.g

isBadVersion result -> FFTTTT

index start from 1  -> 123456

Find right most 'T'

while(lo < hi) -> ending condition lo == hi, means lo and hi overlap at same position ... hi ...

                                                                                          lo

Round 1:

lo=1,hi=6 -> mid=lo+(hi-lo)/2+1=1+(6-1)/2+1=4

isBadVersion(4)=T -> lo=mid=4 (lo=mid means include 'mid')

Round 2:

lo=4,hi=6 -> mid=lo+(hi-lo)/2+1=4+(6-4)/2+1=6

isBadVersion(6)=T -> lo=mid=6 (lo=mid means include 'mid')

Now lo == hi while loop end, and lo can equal to mid, which means include the final answer,

return either hi or lo is fine

**Solution 3: while(lo + 1 < hi) template**

Not suitable for this template because isBadVersion(mid) as an interactive method provide by platform, not response as inequality, but for while(lo + 1 < hi) template to find the last occurrence of target requires explicit inequality equation such as nums[mid] > target and nums[mid] < target

**Refer to**

[L34.Find First and Last Position of Element in Sorted Array (Ref.L278,L704)](note://D037E718AB784CAE8BECDC53F4451102)

[L704.Binary Search](note://484532A5D2CB4B98A28218674438D5D8)