```
/*你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。

假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。

你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。

示例:

给定 n = 5，并且 version = 4 是第一个错误的版本。

调用 isBadVersion(3) -> false
调用 isBadVersion(5) -> true
调用 isBadVersion(4) -> true

所以，4 是第一个错误的版本。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/first-bad-version
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 第一个错误版本号(mid+1)必须满足: $isBadVersion(mid) == false \&\& isBadVersion(mid+1) == true$
  - 方案1:从头开始遍历,时间复杂度$O(N^2)$
  - 方案2:二分查找法,时间复杂度$O(\log N)$,手稿中只实现了二分法。

二分法: C_Solution

```
// Forward declaration of isBadVersion API.
bool isBadVersion(int version);

int firstBadVersion(int n)
{
    int left    = 0;
    int right   = n-1;
    int mid     = 0;
    int ret_val = -1;
    bool flag_1 = false;
    bool flag_2 = false;

    if(n==1)    /*只有一个版本,那说个球,不需要二分,直接处理*/
    {
        return 1;
    }
```

```
    if(n<1) /*不符合题设定义*/
    {
        return -1;
    }

    while(left<=right)
    {
        mid     =   left+(right-left)/2;    /* 防加法溢出*/
        flag_1  =   isBadVersion(mid);
        flag_2  =   isBadVersion(mid+1);

        if(  (flag_1==false)    /*找着啦*/
           &&(flag_2==true)
        )
        {
            ret_val = mid+1;
            break;
        }

        if(  (flag_1==false)    /*还处在全部正确版本的区域*/
           &&(flag_2==false)
        )
        {
            left = mid + 1;
        }

        if(flag_1==true)/*已经处在错误版本的区域*/
        {
            right = mid -1;
        }
    }
    return ret_val;
}
/*
执行结果：
通过
显示详情
执行用时 :4 ms, 在所有 C 提交中击败了85.64% 的用户
内存消耗 :6.7 MB, 在所有 C 提交中击败了7.24%的用户
*/
```

二分法优化逻辑:$mid+1$的版本的正确性不必每次都计算。

```
// Forward declaration of isBadVersion API.
bool isBadVersion(int version);

int firstBadVersion(int n) {
```

```
    int left  = 0;
    int right = n-1;
    int mid   = 0;
    int ret_val = -1;
    bool flag_1 = false;
    bool flag_2 = false;


    if(n==1)
    {
        return 1;
    }

    if(n<1) /*不符合题设定义*/
    {
        return -1;
    }

    while(left<=right)
    {
        mid = left+(right-left)/2;
        if(isBadVersion(mid)==false)
        {
            if(isBadVersion(mid+1)==true)
            {
                ret_val = mid+1;
                break;
            }
            else
            {
                left = mid + 1;
            }

        }
        else
        {
            right = mid - 1;
        }
    }
    return ret_val;
}

/*
执行结果：
通过
显示详情
执行用时 :0 ms, 在所有 C 提交中击败了100.00% 的用户
内存消耗 :6.6 MB, 在所有 C 提交中击败了58.55%的用户
*/
```

AlimyBreak
2019.07.18