题目：

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

示例 1：

```
输入：[2,3,1,1,4]
输出：true
解释：我们可以先跳 1 步，从位置 0 到达 位置 1，然后再从位置 1 跳 3 步到达最后一个位置。
```

示例 2：

```
输入：[3,2,1,0,4]
输出：false
解释：无论怎样，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 ， 所以你永远不可能到达最后一个位置。
```

思路：

只需要 i + nums[i] >= y，则y是可以到达的

所以，遍历数组，更新最大可到达的距离，最大可到达距离 >= n-1 说明就要到达了

```python
def canJump(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        # 只需要 i + nums[i] >= y，则y是可以到达的
        # 所以，遍历数组，更新最大可到达的距离，最大可到达距离 >= n-1 说明就要到达了
        n,rightmost = len(nums), 0
        for i in range(n):
            # 注意，这个判断条件我没考虑，就是如例2，最大可到达的距离是指向0的位置，
            if i <= rightmost:
                rightmost = max(rightmost, i + nums[i])
                if rightmost >= n - 1:
                    return True
        return False
```

跳跃游戏II：

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

你的目标是使用最少的跳跃次数到达数组的最后一个位置。

示例:

```
输入: [2,3,1,1,4]
输出: 2
解释: 跳到最后一个位置的最小跳跃数是 2。
     从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。
```

说明:

假设你总是可以到达数组的最后一个位置。

思路：

一到边界，更新边界，就把步数 + 1（第一步跳到边界以内，再跳一次，一定在边界以外）

那么边界怎么定?

i+nums[i]是能跳的位置，所以就是找在起跳，到边界，这部分能跳的最大位置。

eg：

2，3，1，1，4

起跳是2, 能跳到3， 1的位置，边界就是1的位置2

算新的边界，就是从3开始跳，和1开始跳能到的最远位置：

3 ---> 4 (1--->4 i+nums[i])

1 ---> 1 (2 --->3 i+nums[i])

所以接下来新的边界应该是4的位置，4

```python
def jump(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n = len(nums)
        maxPos, end, step = 0, 0, 0
        # end: 当前跳的边界
        # maxPox: 跳到最远的位置
         # step: 跳了几步
         for i in range(n - 1):
             if maxPos >= i:
                 maxPos = max(maxPos, i + nums[i])
                 if i == end:
                     end = maxPos
                     step += 1
         return step
```

时间复杂度O(n)

空间复杂度O(1)