55. Jump Game

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the last index, or false otherwise.

Example 1:

Input: nums = [2,3,1,1,4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

SOLUTION

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int terminal = nums.size() - 1;
        int m = 0;      // max value to arrive
        for (int i=0; i<=terminal; i++) {
            if (i <= m) {   // possible to arrive
                m = max(m, i+nums[i]);  // refresh right most point
                if (m >= terminal)
                    return true;        // access the terminal
            }
        }
        return false;
    }
};
```

45. Jump Game II

You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach* nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:

Input: nums = [2,3,1,1,4] **Output:** 2

Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2: Input: nums = [2,3,0,1,4] **Output:** 2

SOLUTION

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int terminal = nums.size() - 1;
        int steps = 0, left = 0, right = 1;

        while (right <= terminal) {
            int m = 0;
            for (int i=left; i<right; i++) {
                m = max(m, i+nums[i]);
            }
            left = right;
            right = m+1;
            steps++;
        }
        return steps;
    }
};
```