

`nums[]` — 5 elements, need to reach index 4

2	3	1	1	4
---	---	---	---	---

Init: jumps=0, curEnd=0, farthest=0

Step 1 i=0: farthest=max(0,0+2)=2. i==curEnd(0) → jump! jumps=1, curEnd=2

2	3	1	1	4
---	---	---	---	---

Step 2 i=1: farthest=max(2,1+3)=4. i≠curEnd(2) → no jump yet

2	3	1	1	4
---	---	---	---	---

Step 3 i=2: farthest=max(4,2+1)=4. i==curEnd(2) → jump! jumps=2, curEnd=4

2	3	1	1	4
---	---	---	---	---

Step 4 i=3: farthest=max(4,3+1)=4. i≠curEnd(4) → no jump

2	3	1	1	4
---	---	---	---	---

i=4 = last index, stop loop. Answer = 2 jumps ✓

■ Java Code

```
public static int jump(int[] nums) {
    int jumps = 0; // Total jumps taken
    int curEnd = 0; // End of current jump's window
    int farthest = 0; // Farthest reachable in this window
    // Don't process last element (already there if we reach it)
    for (int i = 0; i < nums.length - 1; i++) {
        farthest = Math.max(farthest, i + nums[i]);
        if (i == curEnd) {
            // We've exhausted current window – must jump
            jumps++;
            curEnd = farthest;
        }
    }
    return jumps;
}
```

■■ Time & Space Complexity

Case	Time	Space	What it really means
------	------	-------	----------------------

All Cases

 $O(n)$ $O(1)$

One loop, three integer variables — nothing else

■ Jump I vs Jump II

- Jump I: can you reach? → simple maxReach check → return boolean
- Jump II: min jumps to reach? → window-based greedy → return count
- Both are $O(n)$ time, $O(1)$ space — same elegance, different goals

✓ Greedy Approach

- $O(n)$ time, $O(1)$ space
- Track 3 variables
- Extend window only when forced
- Clean single loop

✗ BFS / DP

- $O(n)$ time, $O(n)$ space
- Needs queue or dp array
- More code, same result
- Harder to reason about

■ Common Mistakes

- Loop should go to `nums.length - 1` (not `nums.length`) — last index needs no jump from it
- Forgetting that jumping happens when `i == curEnd`, not `farthest`
- Confusing `curEnd` (window end) with `farthest` (best reachable in window)

■ Visual Thinking

Imagine a frog leaping across lily pads in a pond.

From each pad, the number tells you how far your legs can stretch.

You want to reach the far bank (last index) in as few leaps as possible.

Strategy: from your current pad, look at every reachable pad — pick the one that gets you farthest NEXT.

Only leap when you're about to fall off the current pad's reach.

Animation: frog stays on pad, scanning forward. Dashes show reach. Leap happens at window boundary.

■ Practice Problems

[Easy] Trace `[1,2,1,1,1]` — how many jumps?

[Med] What if `nums=[1,1,1,1,1]`? How many jumps for `n` elements?

[Med] Jump Game II with obstacles (some cells impassable)

[Hard] Jump Game II where jump cost = `nums[i]` — minimize total cost

■ Quick Revision Box

- 3 variables: `jumps`, `curEnd`, `farthest`
- At each `i`: update `farthest = max(farthest, i + nums[i])`
- When `i == curEnd`: `jump!`, `jumps++`, `curEnd = farthest`
- Loop to `nums.length - 1` (not `length`)
- $O(n)$ time, $O(1)$ space
- Window-based greedy — extend only when forced