

House Robber II

Circular street — break the circle cleverly ■

● MEDIUM

■ Hook

Same as House Robber, but now houses are arranged in a **circle**. So house 0 and house n-1 are neighbors — you can't rob both. Trick: run House Robber I twice — once excluding last house, once excluding first. Take max.

■ Core Intuition — Break the Circle

■ The Key Insight

In a circle, house[0] and house[n-1] can't both be robbed. So the optimal solution either **INCLUDES** house[0] (must exclude house[n-1]) OR **INCLUDES** house[n-1] (must exclude house[0]). Run Linear House Robber on [0..n-2] and [1..n-1]. Answer = max of both.

■ Visual — [2,3,2]

```
nums = [2, 3, 2] (circular: house0 and house2 are adjacent)
Run 1: exclude last → rob [2,3] → max=3
Run 2: exclude first → rob [3,2] → max=3
ANSWER = max(3,3) = 3

nums = [1,2,3,1]
Run 1: [1,2,3] → dp = 1,2,4 → max=4
Run 2: [2,3,1] → dp = 2,3,3 → max=3
ANSWER = max(4,3) = 4 ✓ (rob house0=1 and house2=3)
```

■ Code

```

public int rob(int[] nums) {
    if (nums.length == 1) return nums[0];
    if (nums.length == 2) return Math.max(nums[0], nums[1]);
    return Math.max(
        robLinear(nums, 0, nums.length-2), // exclude last
        robLinear(nums, 1, nums.length-1) // exclude first
    );
}
private int robLinear(int[] nums, int lo, int hi) {
    int prev2 = 0, prev1 = 0;
    for (int i = lo; i <= hi; i++) {
        int curr = Math.max(prev1, prev2 + nums[i]);
        prev2 = prev1; prev1 = curr;
    }
    return prev1;
}

```

■ TIME

$O(n)$

■ SPACE

$O(1)$

- ✓ Circular → can't rob first AND last
- ✓ Split into two linear subproblems: $[0..n-2]$ and $[1..n-1]$
- ✓ Answer = max of both runs
- ✓ Reuse House Robber I logic exactly
- ✓ Time: $O(n)$ | Space: $O(1)$

■ MENTAL MODEL — Remember This Forever

Breaking a circle into two lines is a universal technique in circular DP. Whenever you have circular constraints, fix one element's status (include/exclude) and solve the remaining linear problem. Run both versions, take the best.