

# House Robber

## Example 1:

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

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob =  $1 + 3 = 4$ .

## Example 2:

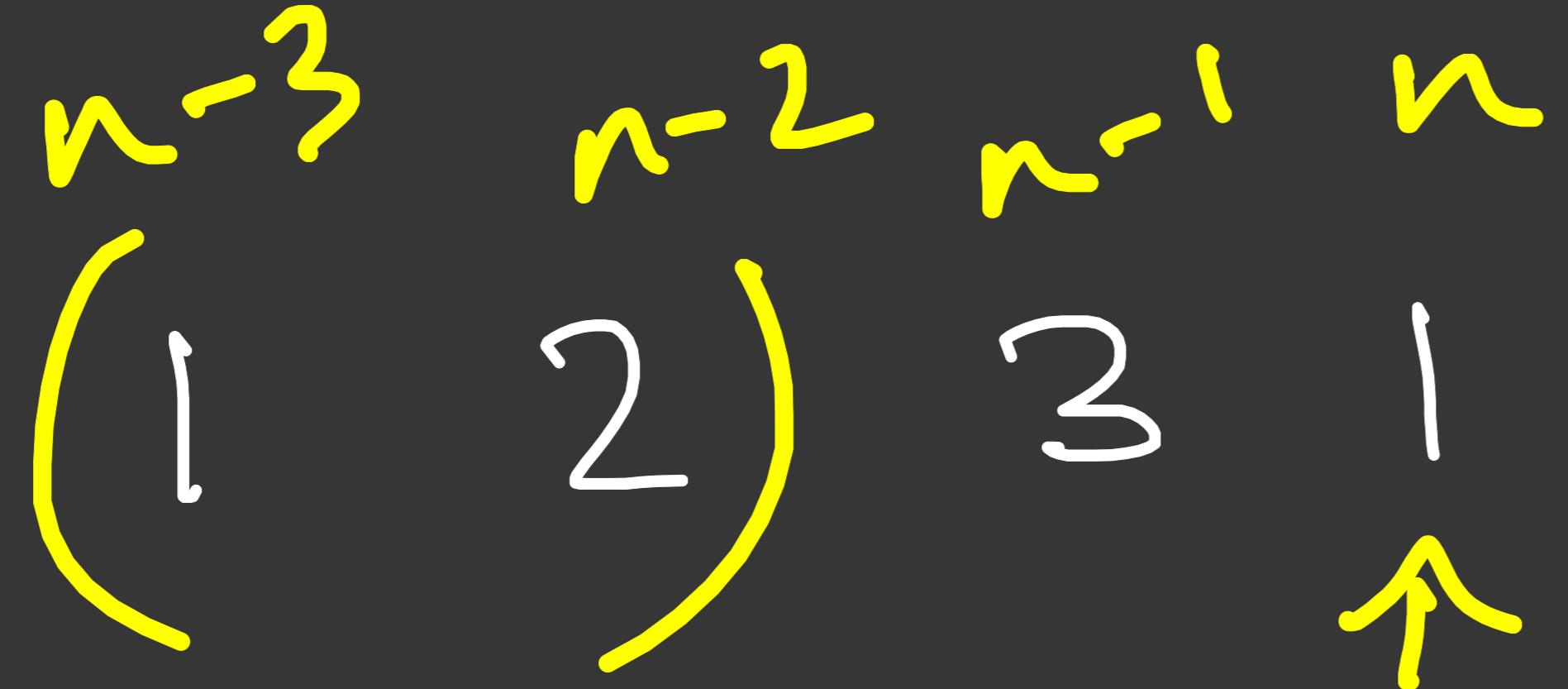
**Input:** nums = [2,7,9,3,1]

**Output:** 12

**Explanation:** Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob =  $2 + 9 + 1 = 12$ .

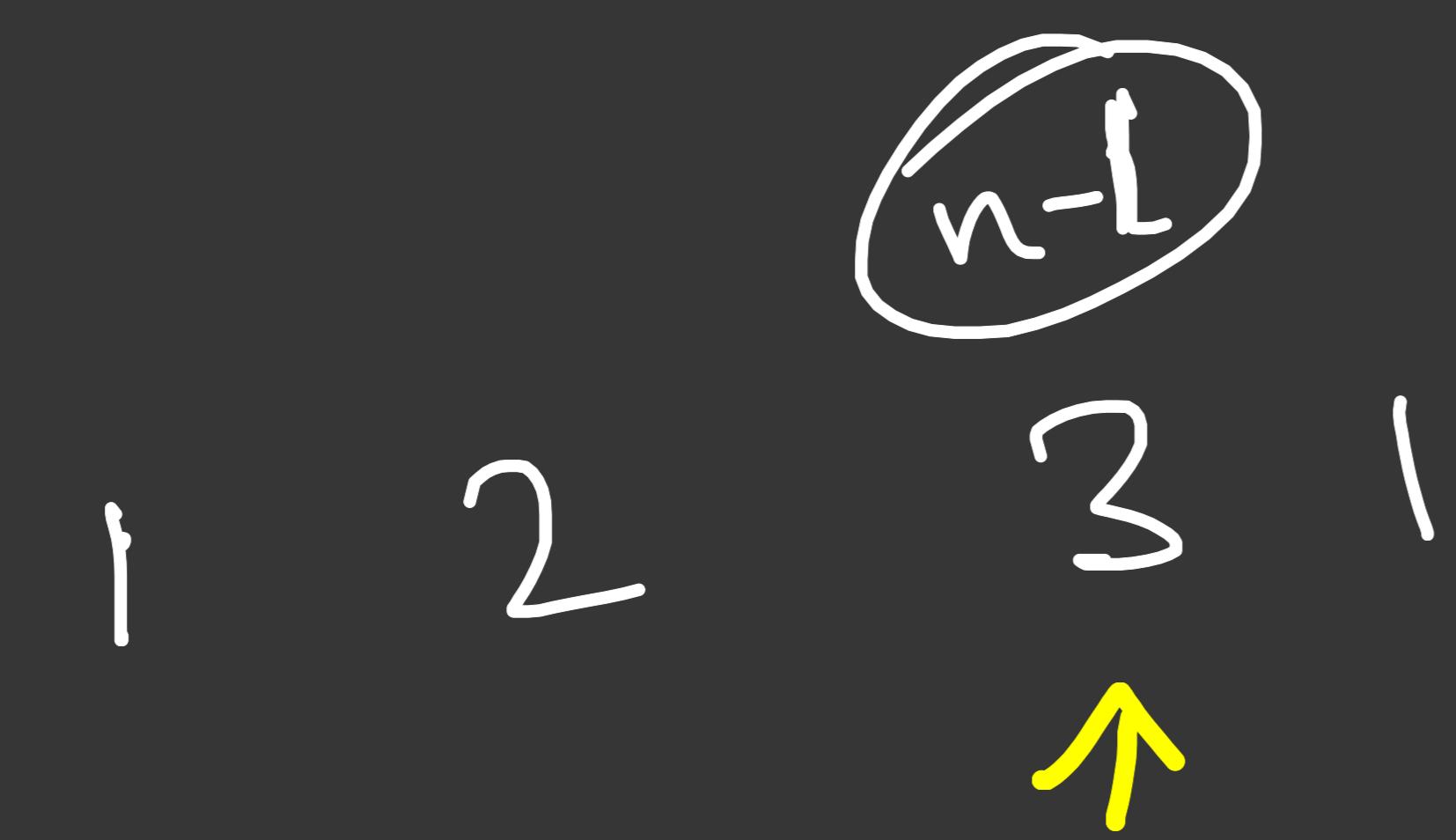
Case 1  
arr  $\Rightarrow$



$$\text{arr}[i] + \text{Rob}(\text{arr}[i-2])$$

Case 1:- (Yes)

Case 2  
arr  $\Rightarrow$



$$\boxed{\text{arr}[i-1]}$$

if ( $i \leq 0$ )  
return 0;  
(means there is no house)

Case 2:- (No)

int ith\_house\_is\_selected = rob(arr[i] + rob(i-2));  
int i'm house is not selected = rob(i-1);  
return max(selected, notSelected);

## Memoization Solution

```
int Rob (arr, i, dp) {  
    if (i < 0) return 0;  
    if (dp[i] != INT_MIN)  
        return dp[i];  
    int robith = arr[i] + Rob (arr, i - 2, dp);  
    int notRobith = Rob (arr, i - 1, dp);  
    return max (robith, notRobith);  
}
```

y

## Recursive Sol<sup>n</sup>

```
int rob (vector<int>& wealth, n) {  
    if (n == 0) return 0;  
    int subith = arr[n] + rob (wealth, n - 1);  
    int notrobith = rob (wealth, n - 1);  
    return max (subith, notrobith);  
}
```

```
int Rob (vector<int>& nums) {  
    int n = nums.size();  
    return rob (nums, n - 1);  
}
```

## Top down Approach

```
int Rob(vector<int>& nums) {  
    int n = nums.size();  
    vector<int> dp(n+1);  
    dp[0] = 0; dp[1] = 1;  
    for (int i = 0; i <
```





























































































































































































