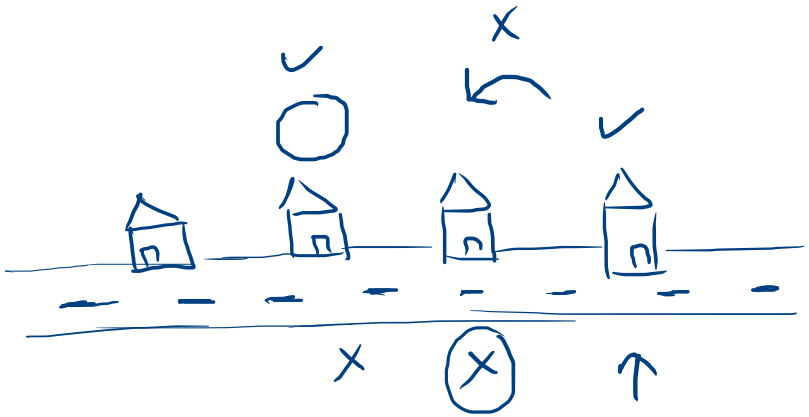
House Robber

I/p: nums = [1, 2, 3, 1] (n=4)

O/p: 4

(1 2 3 1) → 3



I/p : nums = [2, 7, 9, 3, 1]

O/p : 12

7 + 3 = 10

[2 7 9 3 1] = 12

Recursive Solution

I/p → **Magical Function** → o/p

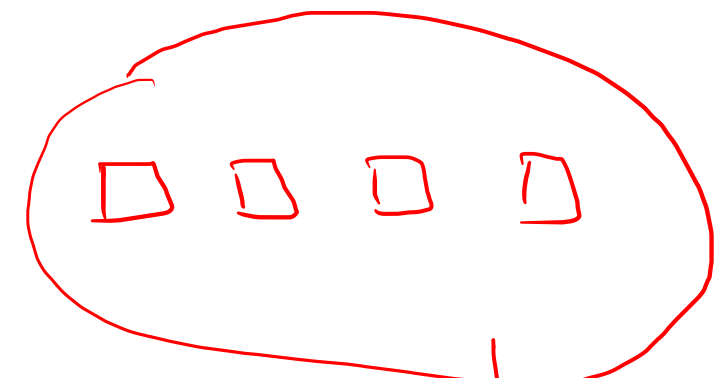
(nums[], int n)

- nums[] → money in each house
- int n → Number of house consider to be stolen

o/p → max^m amount of money stolen from house index 0 to n-1

nums[]

n=4

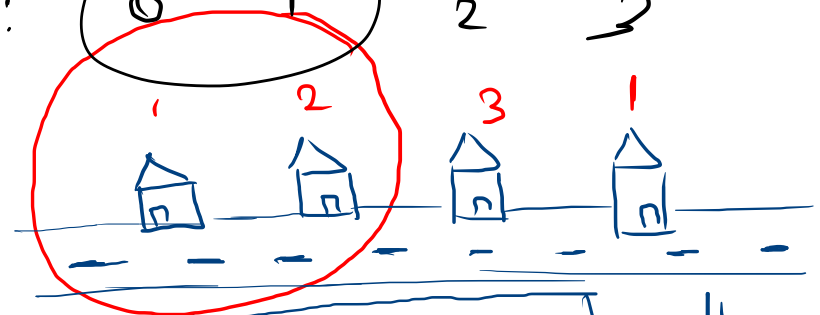


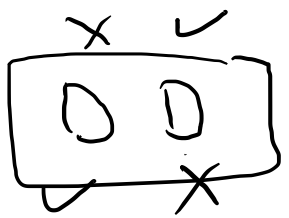
fun(nums, 4)

Recursive Solution

I/p → (nums[], int n) → Magical Function → O/p → max^m amount of money stolen from House index 0 to n-1

index : 0 1 2 3





nums = [1, 2, 3, 1]

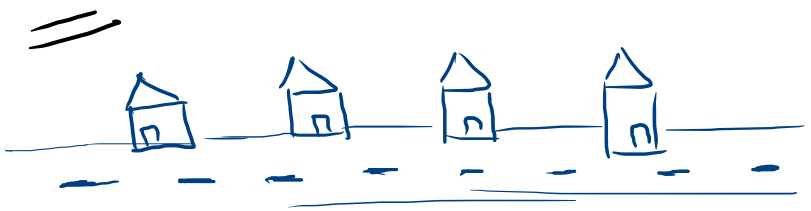
n = 2

→ fun( ) → O/p → max^m amount can be stolen from House index 0 to 1 = 2

Recursive Solution

I/p → **Magical Function** → O/p

(nums[], int n) → max^{m} amount of money stolen from House index 0 to n-1



↳ ultimately we will call,

maximum Amount ← fun(nums, n)

I/p n = 4

nums[] = {1, 2, 3, 1}

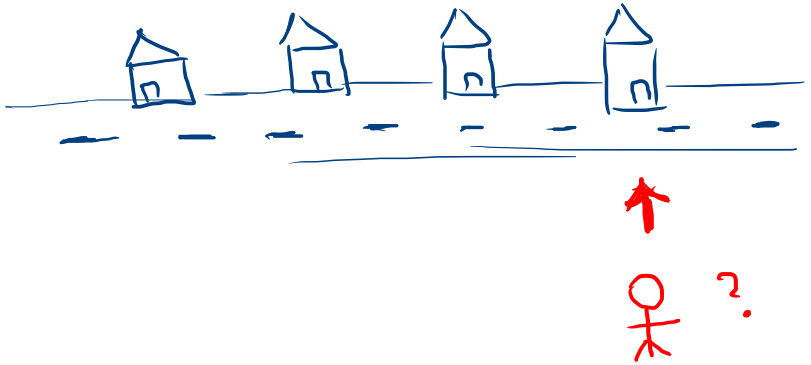
fun(nums, 4) → max^{m} amt.

Recursive solution

** Logical part

I/p: [1, 2, 3, 1]

O/p: n = 4



thief has 2 option

do

don't

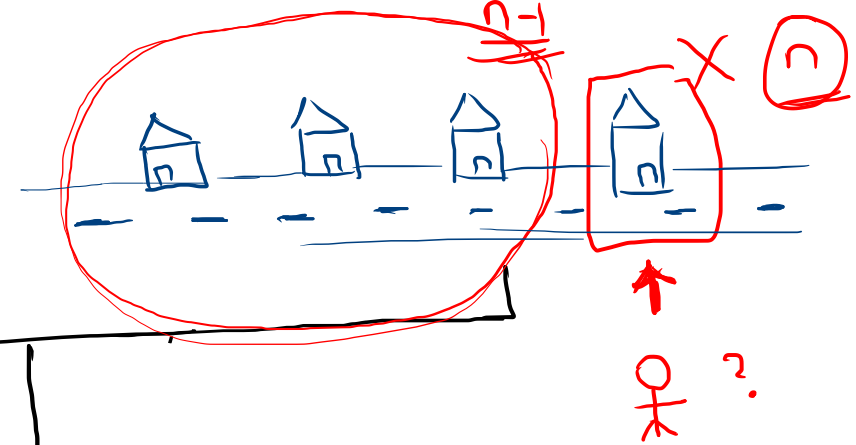
Recursive solution

Logical part ☺

I/p: [1, 2, 3, 1]

O/p:

He will not choose the last House



thief has 2 option

do

don't

amount1 = fun(nums[] n-1);

↳ Consider n-1 House

Recursive solution

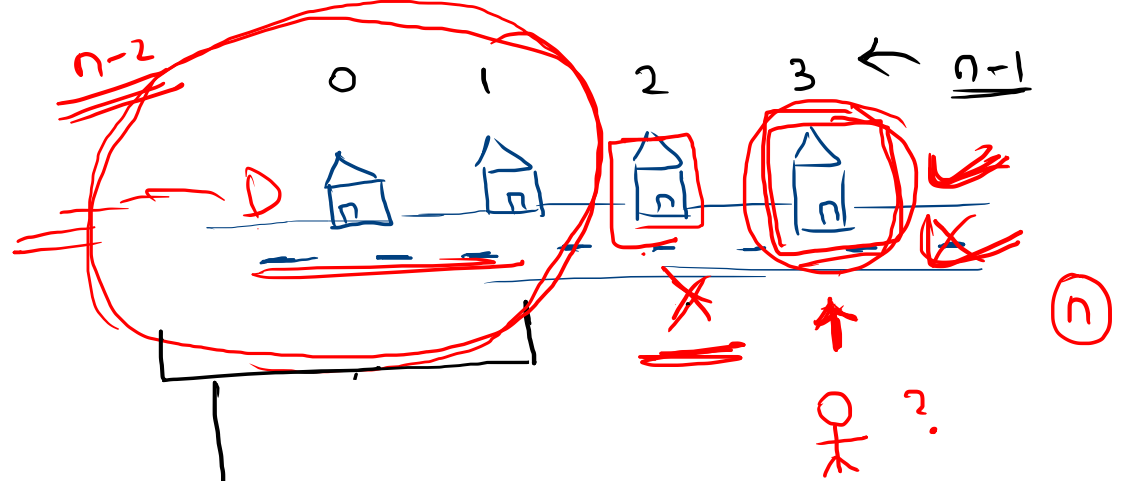
Logical part ☺

I/p: [1, 2, 3, 1] nums[]

O/p:

He will choose the last House

0 1 2 3 .... n-1



thief has 2 option

do don't

amount2 = nums[n-1] + fun(nums[], n-2);

Stolen House amount

Consider n-2 House

Recursive solution

Logical part ☺

I/p: [1, 2, 3, 1]

O/p:

So, thief will choose the maximum amount out of his 2 choices

thief has 2 option

do don't

amount = max (fun(nums, n-1) , nums[n-1] + fun(nums, n-2)) ;

Recursive code

```
int fun ( nums[ ] , int n)
{
    // Base Condition

    return max ( fun(nums, n-1) ,
                 nums[n-1] + fun (nums, n-2)) ;
}
```

Base condition

Smallest Valid Input → check output

int fun (nums [ ] , int n)

n = 0, 1, 2, 3 .....

nums [ ] = { } amount = 0

amount = 0

Recursive code

```
int fun ( nums[] , int n)
{
    if ( n <= 0) return 0;
    return max ( fun(nums, n-1) ,
                 nums[n-1] + fun (nums, n-2)) ;
}
```

Base Condition

Logic

TLE

fun(nums, n)

fun(num, n-1)

fun(nums, n-2)

fun(num, n-2)

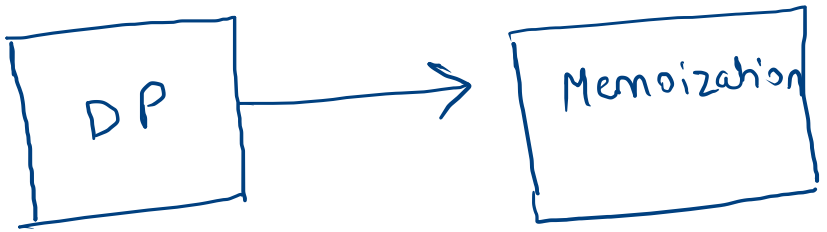
fun(num, n-3)

fun(num, n-3)

f(n-4)

func Recursive

Memoization Code



[vector <int> dp (101, -1) ;]

Recursive code

```
int fun ( nums[] , int n)
{
    if ( n <= 0) return 0;
    [ if ( dp[n] != -1) return dp[n] ; ]
    return dp[n] = max ( fun(nums, n-1) ,
                         nums[n-1] + fun (nums, n-2)) ;
}
```