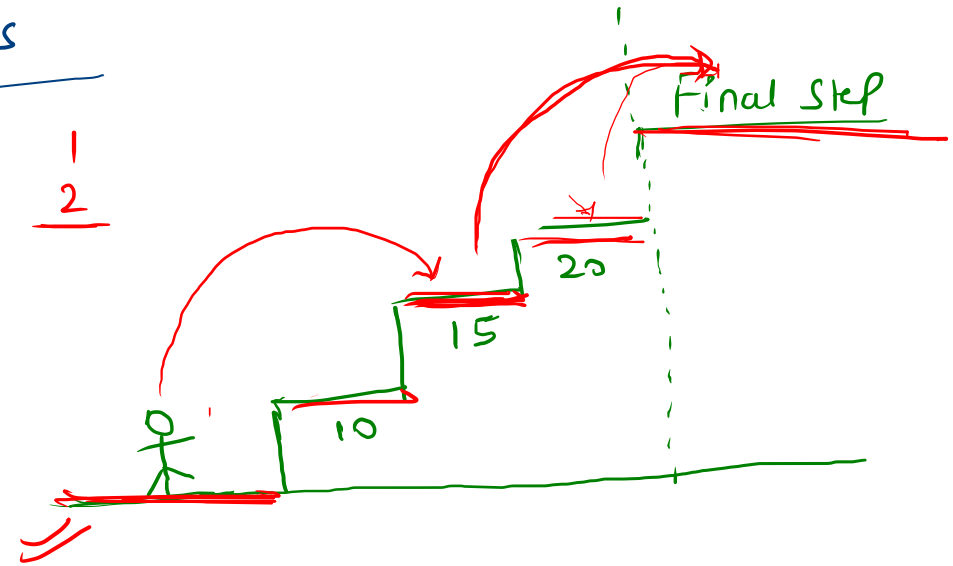
Min Cost Climbing Stairs

I/p: cost[] = {10, 15, 20}

O/p: 15



1

2

10 + 20 = 30

15 = 15

we will try to write Recursive solution for this problem

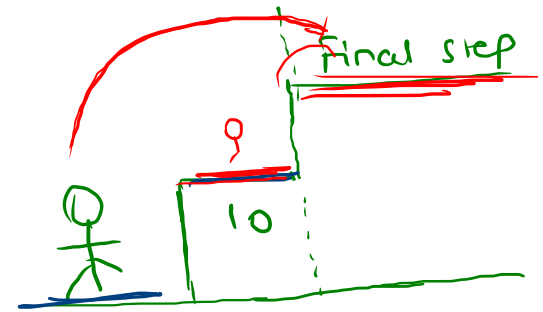
#

Base Condition

I/p: arr[] = {10}

O/p: 0

if(n==1) return 0;



10 + 1 = 11

I/p: arr[] = { }

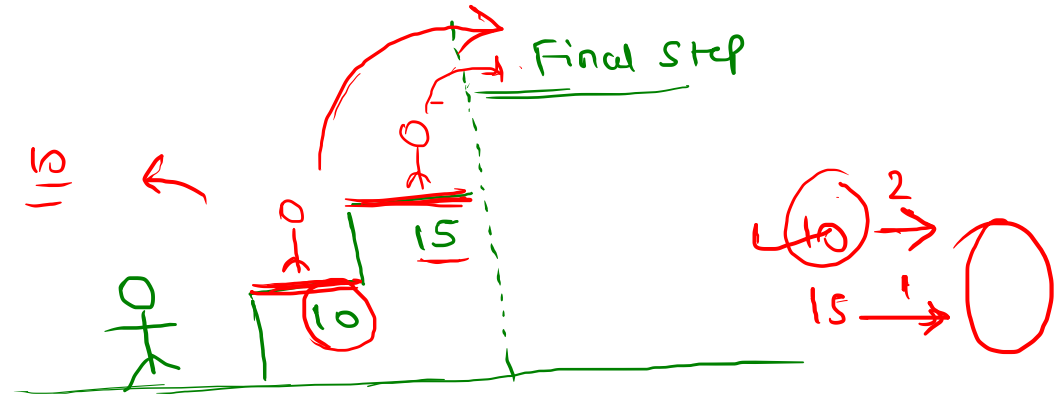
return 0;

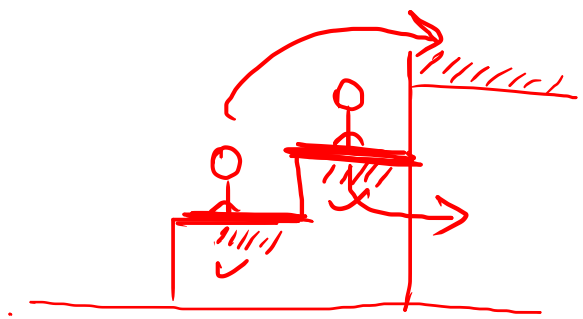
I/p: arr[] = {10, 15}

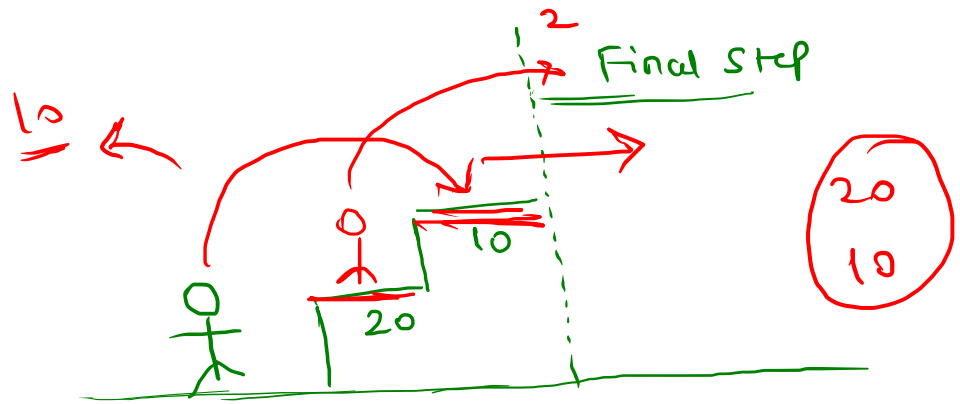
O/p:

cost[0]
cost[1]
] min

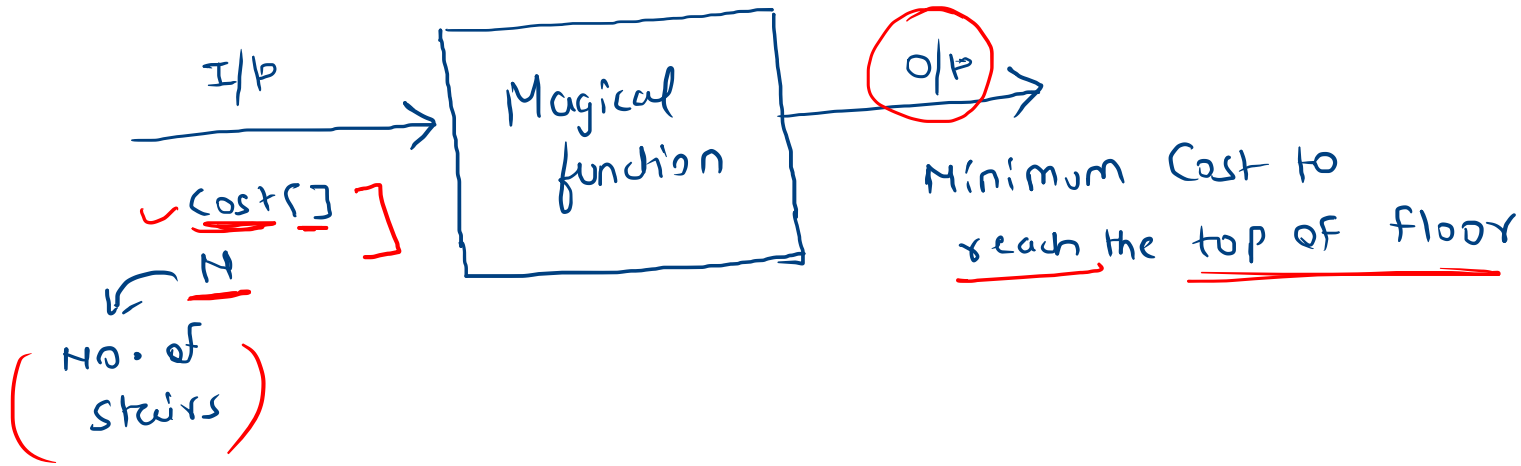
if(n==2)

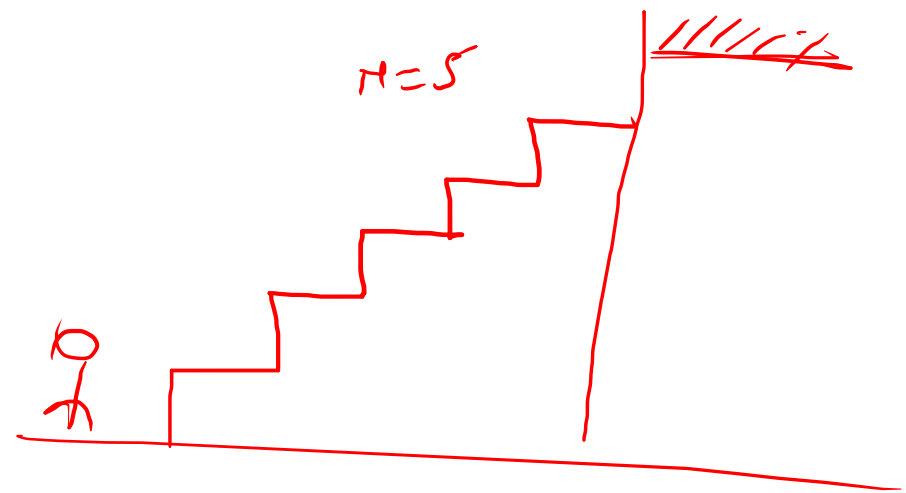




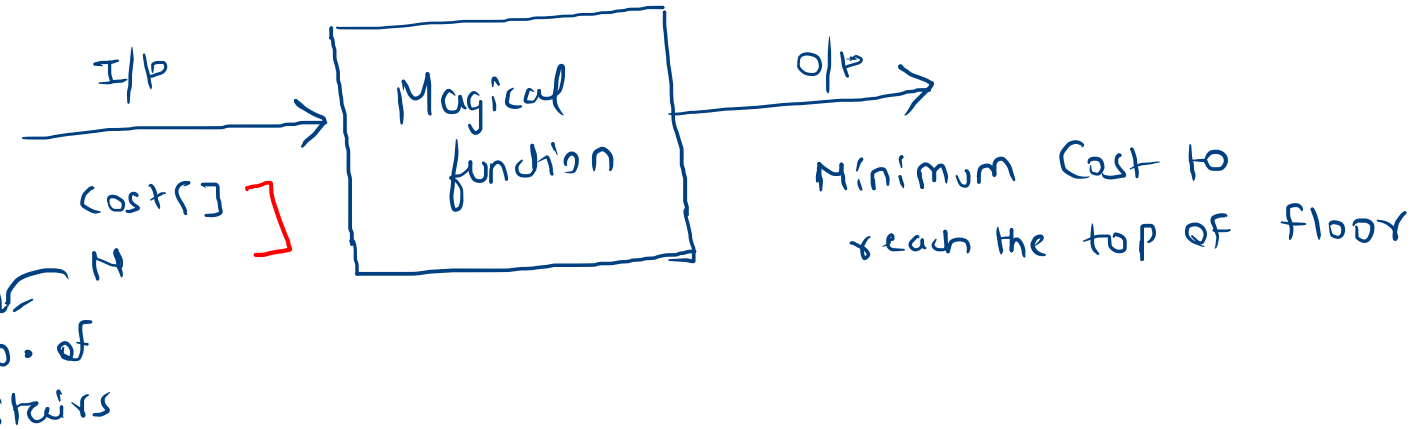


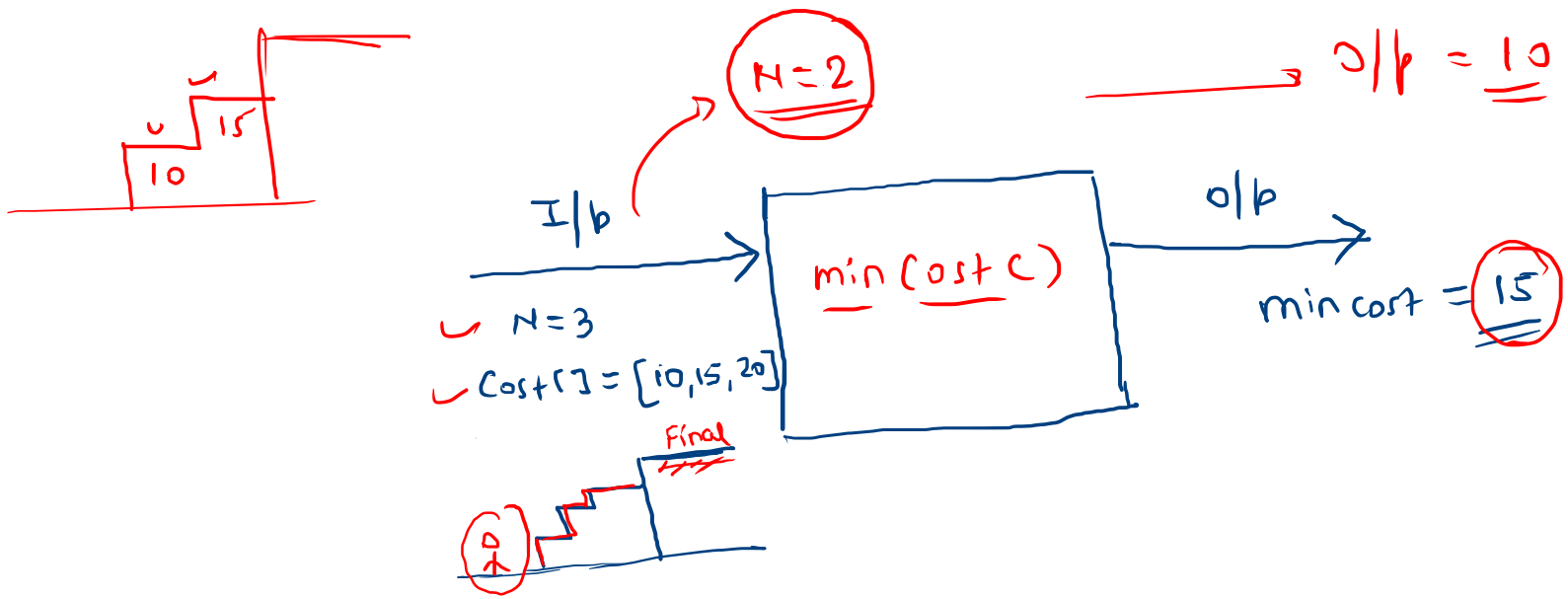
Recursive solution





Recursive solution

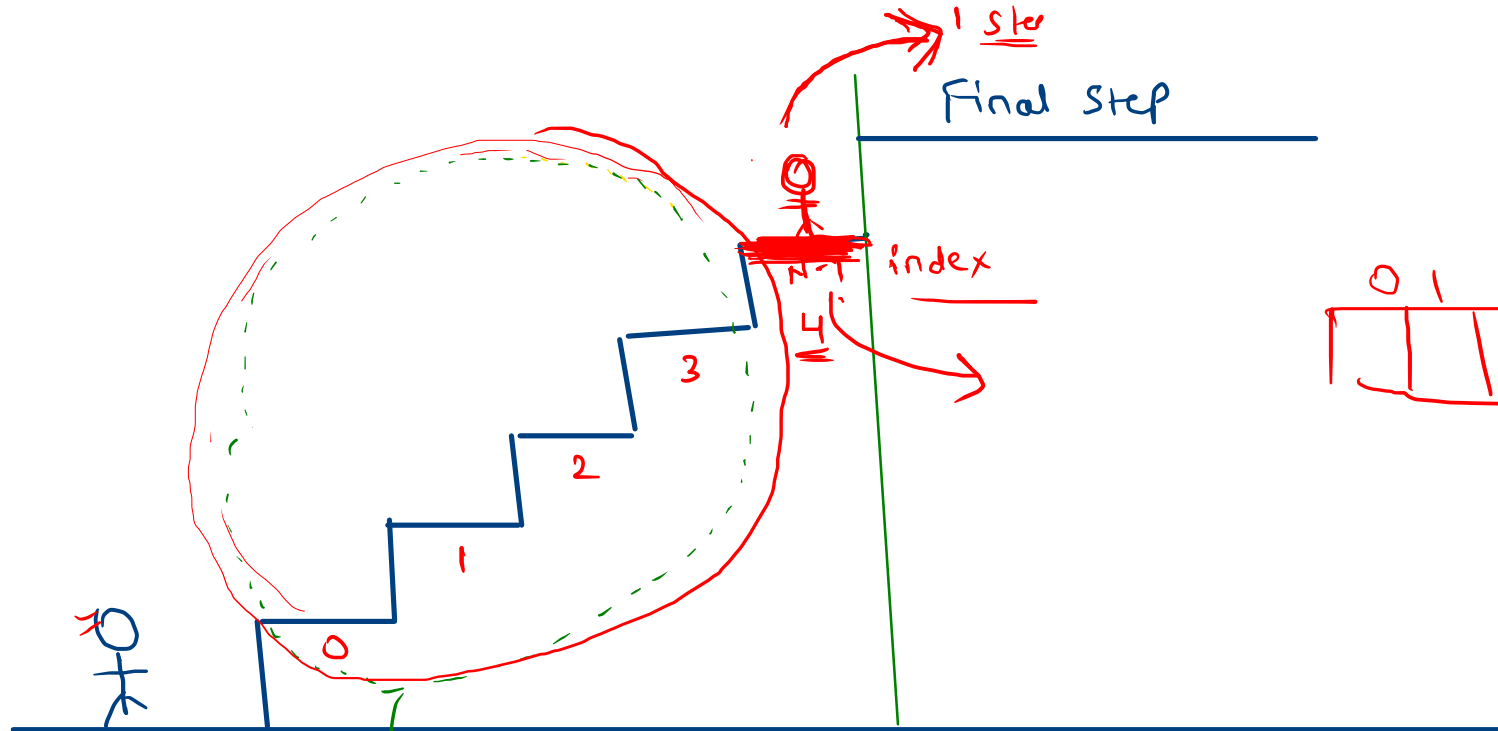




Rewrsive function

Logic

N=5



remaining (N-1) Stairs

N=4

Rewrsive fn

minClimb (cost[] , N-1) → Cost to cover N=4 stairs]

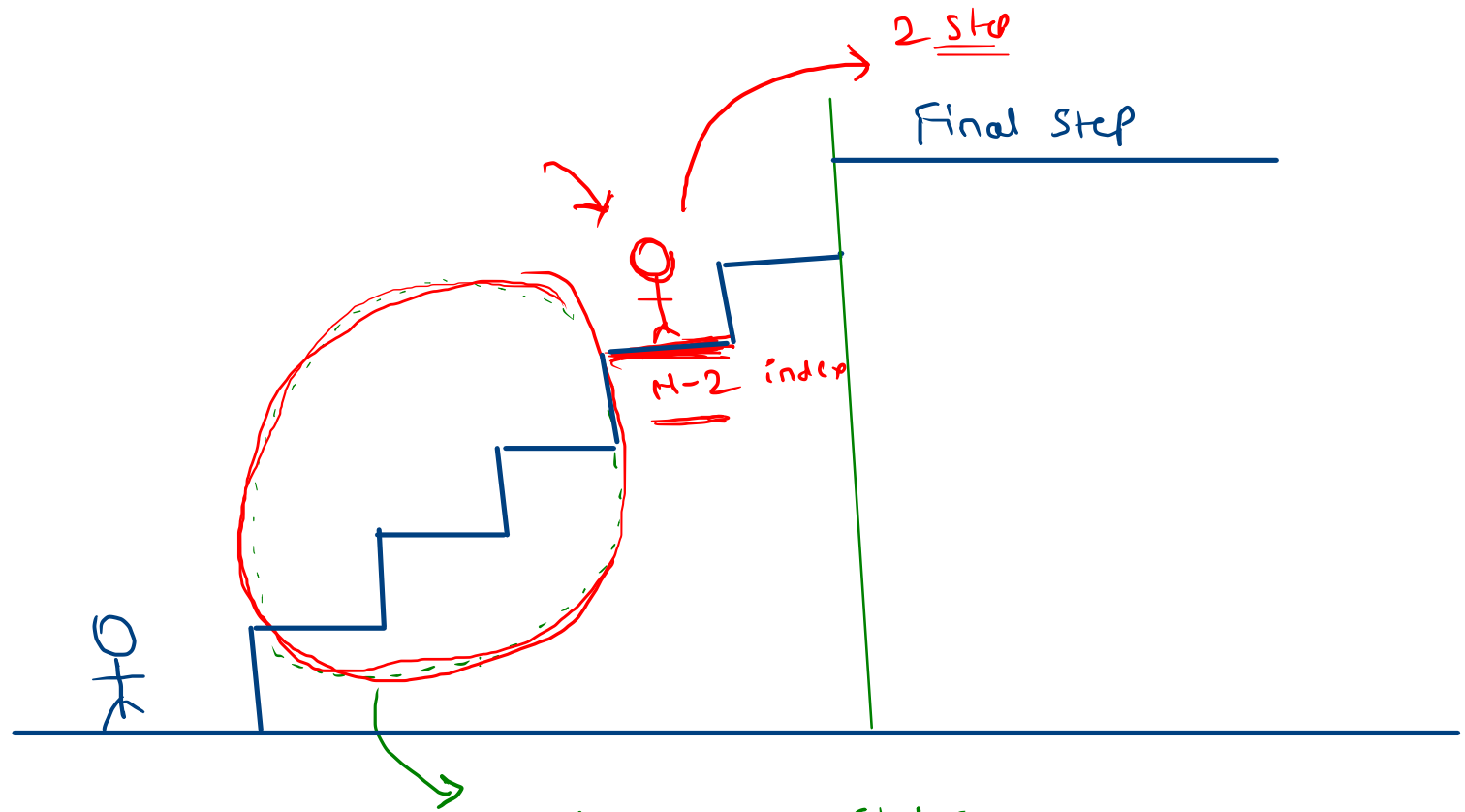
\+

Cost [N-1] → Cost to reach N-1 th step

Recursive function

Logic

N=5



(N=5)

N=3

minClimb (cost[] , N-2)

+

Cost [N-2]

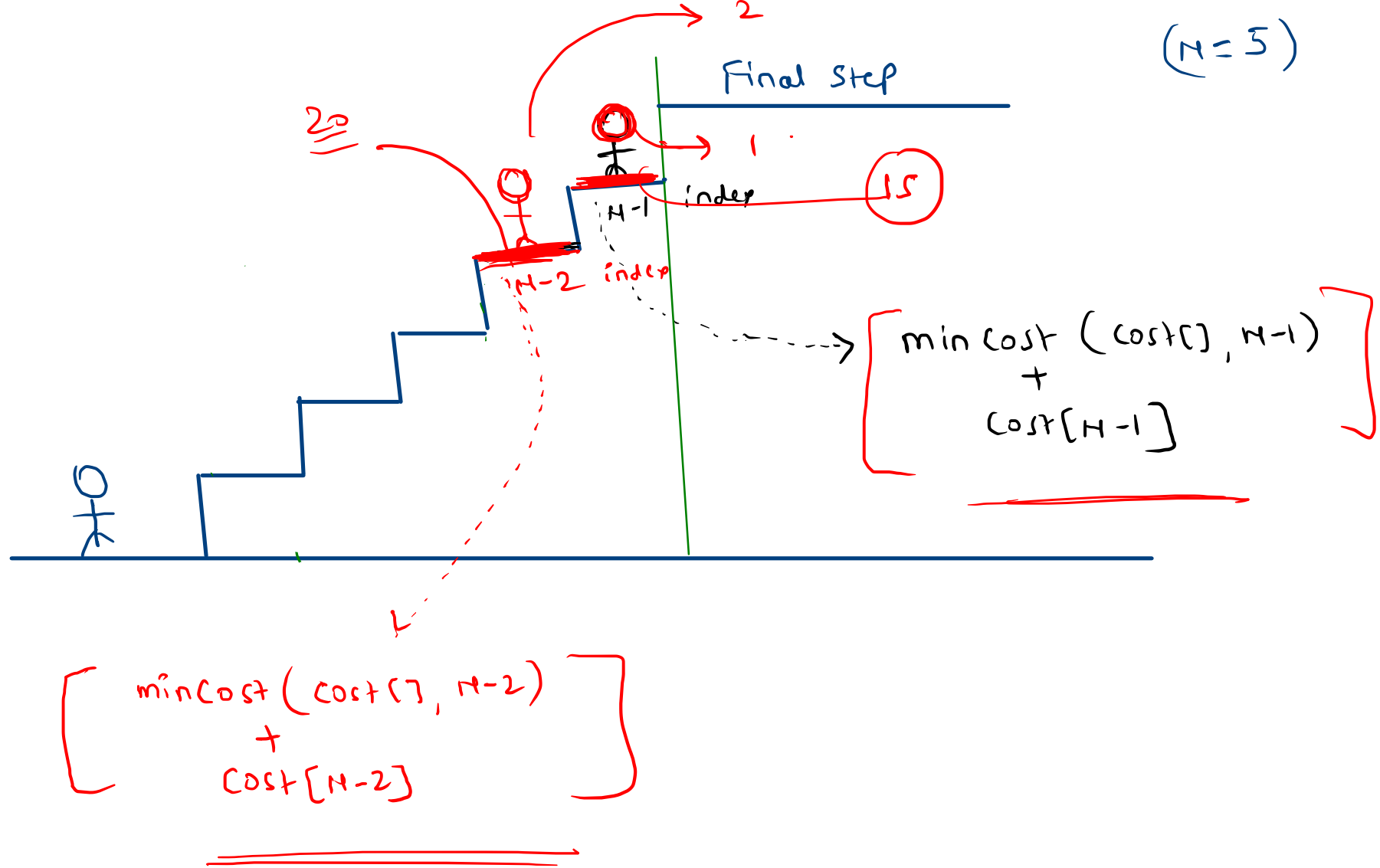
Cost to cover N=3 stairs

Cost to reach N-2th step

Recursive function

Logic

N=5



Recursive solution

```
int minCost(int cost[], int N)        // N -> no. of stairs
{
    // Base condition

    return min( minCost(cost[], N-1) + cost[N-1],
                minCost(cost[], N-2) + cost[N-2] );
}
```

Base condition

+ Smallest valid Input → check output

```
int minCost(int cost[], int N)
```

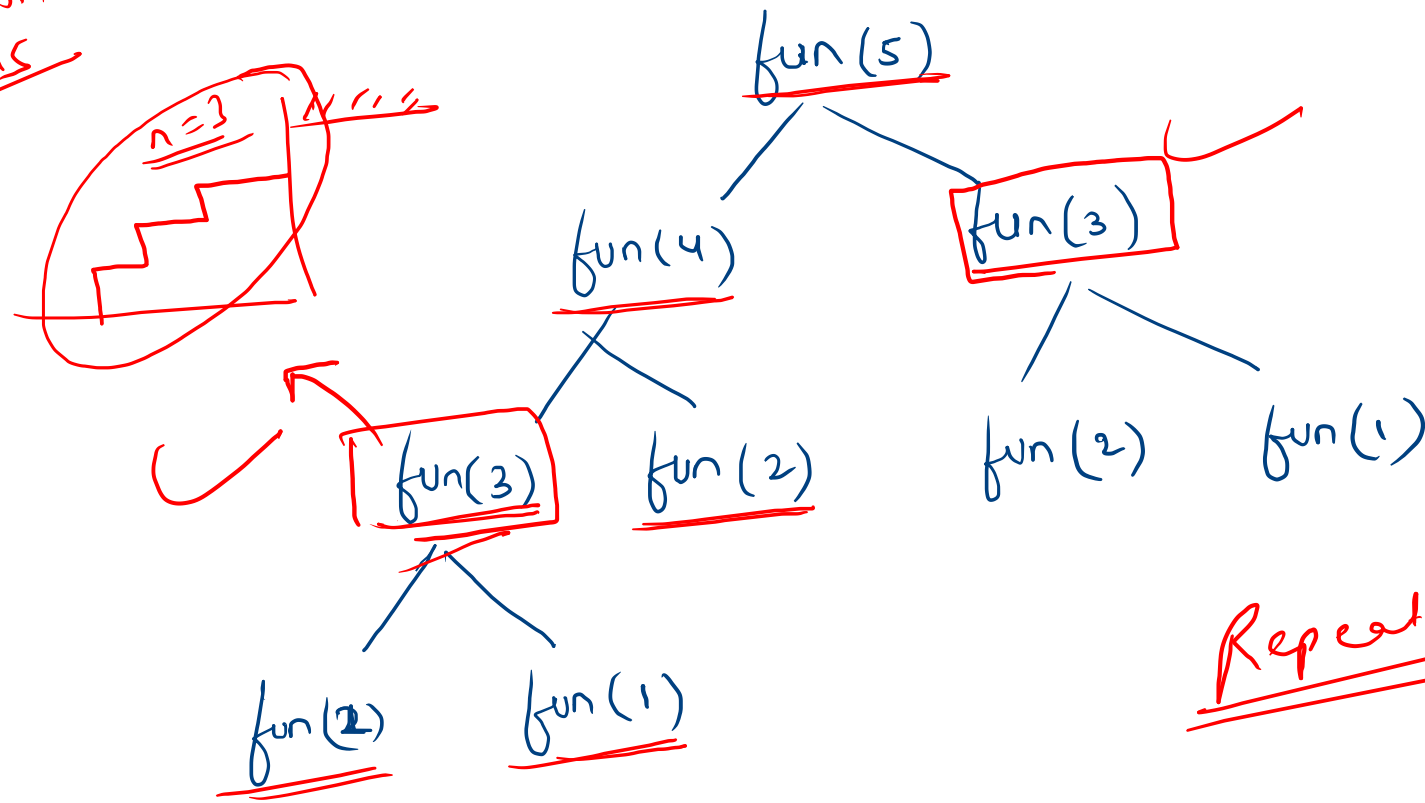
```
if (N==0) return 0;
if (N==1) return 0;
if (N==2) return
        min(cost[0], cost[1]);
```

Recursive solution

```
int minCost(int cost[], int N)        // N -> no. of stairs
{
    // Base condition
    if (N <= 1) return 0;
    if (N == 2) return min(cost[0], cost[1]);

    // Logic
    return min(minCost(cost[], N-1) + cost[N-1],
               minCost(cost[], N-2) + cost[N-2]);
}
```

Function calls



minCost(cost[] ,5)

⇓

fun(5)

Repeat DP ✓

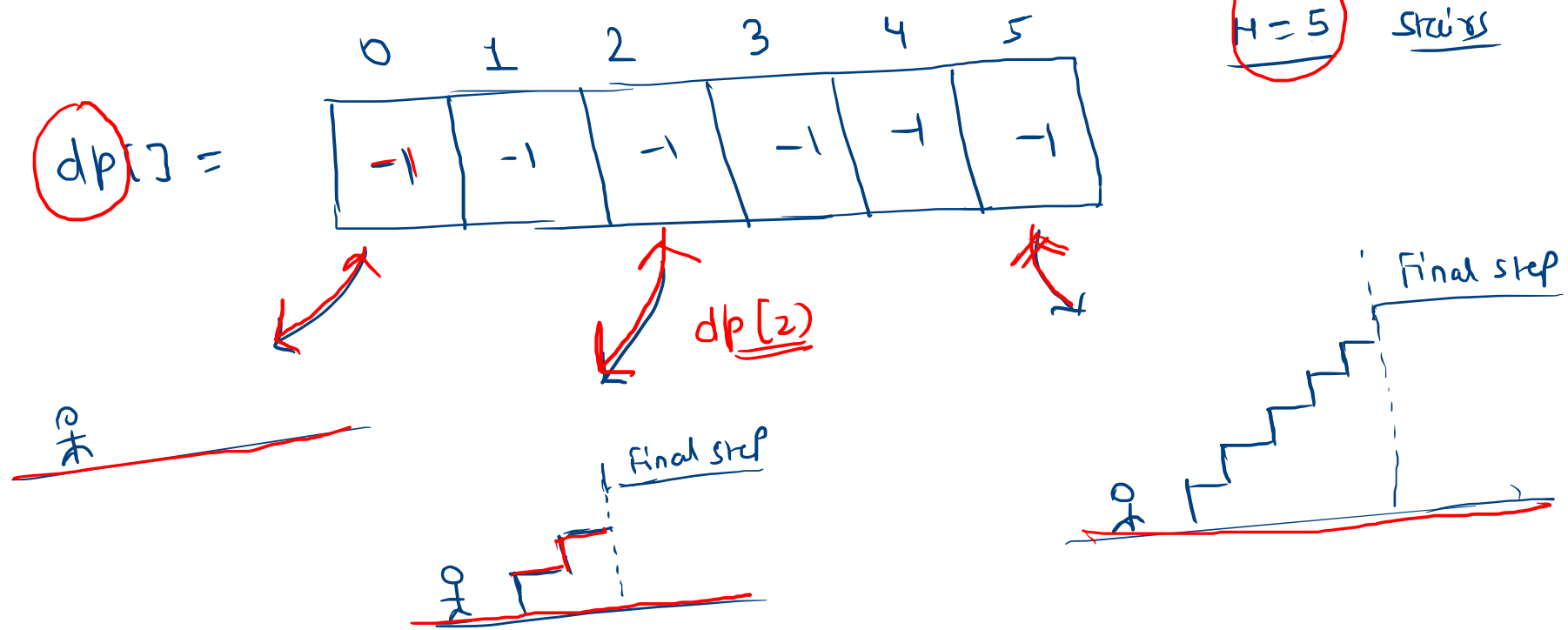
n-1

n-2

Recursive code → DP Code

Memoization

dp(n+1)



Vector <int> dp(1001, -1);

Recursive solution

int minCost (int cost[], int N) ← No. of stairs

{

if (dp[n] != -1) return dp[n];

Base condition [if (N <= 1) return dp[n] = 0;

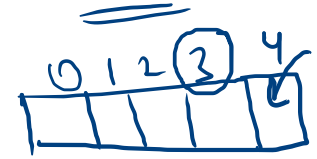
if (N == 2) return dp[n] = min (cost[0], cost[1])

Logic [return dp[n] = min (minCost (cost[], N-1) + cost[N-1],

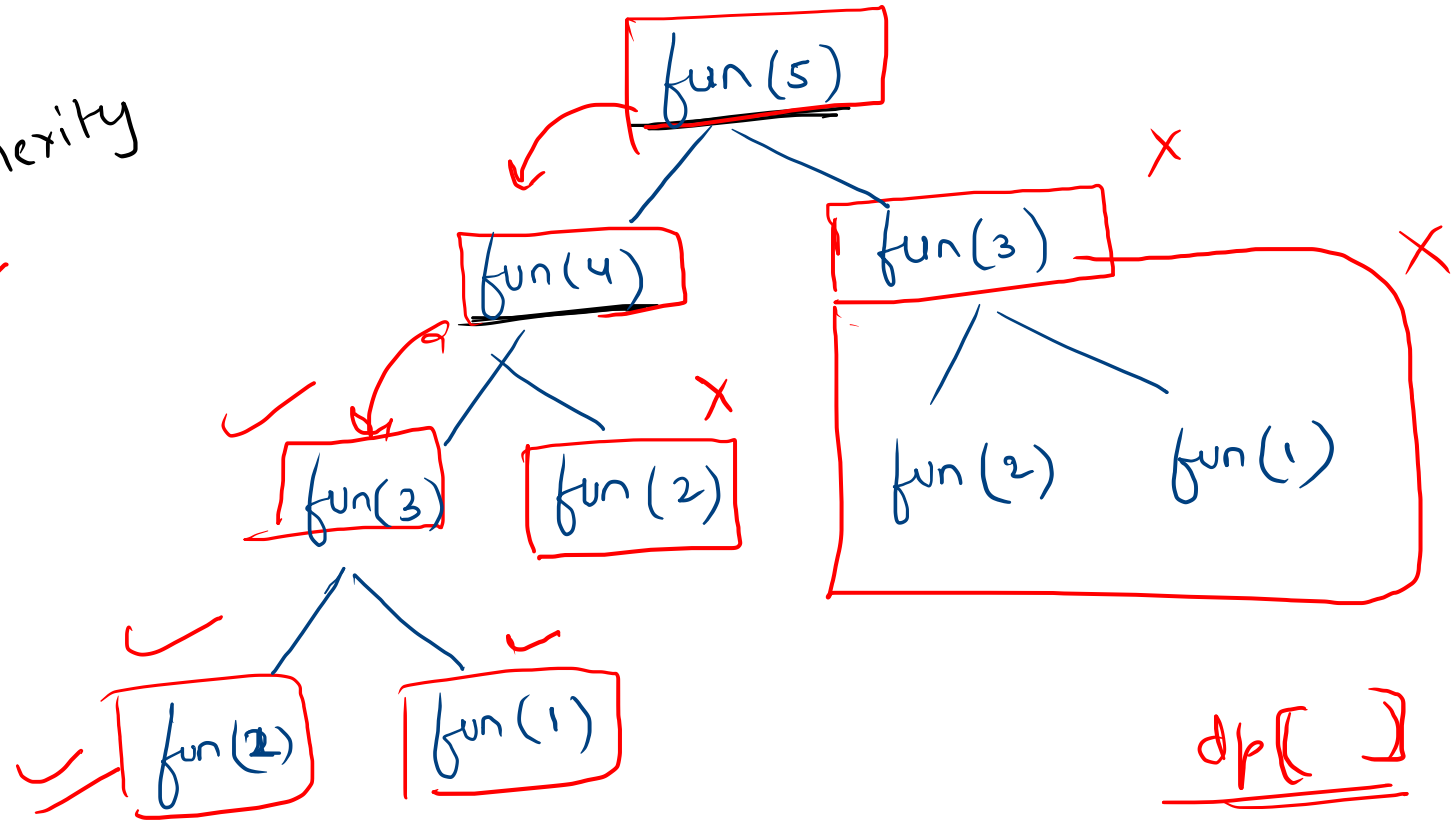
minCost (cost[], N-2) + cost[N-2]);

}

N = 4



Function calls



Time complexity

O(n)

minCost(cost[] ,5)

⇓

fun(5)

dp[ ]

Space complexity

O(n)