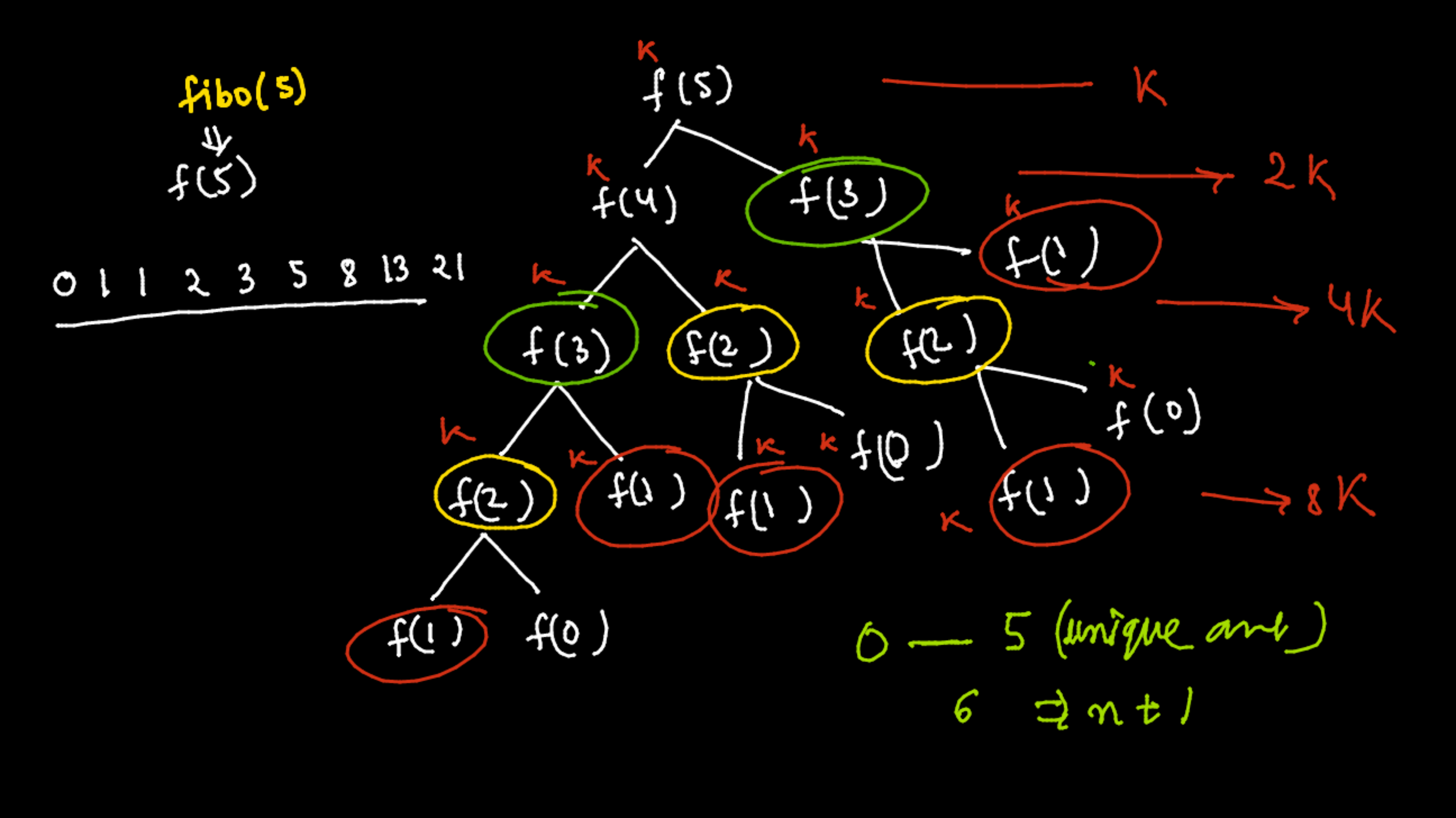
18. Dynamic Programming - I

(i) Fibo using recursion से DP में Memoization

(ii) Min steps to 1 Brute / Memoization / DP

(iii) Min count (of squares) Brute / Memoization / DP

(iv) No. of BT " " "


fibo(5)
f(5)
0 1 1 2 3 5 8 13 21
K
f(5)
f(4)
f(3)
f(3)
f(2)
f(2)
f(1)
f(2)
f(1)
f(0)
f(1)
f(1)
f(0)
f(1)
f(0)
K
2K
4K
8K
0 — 5 (unique ans)
6 ⇒ n+1

DP : (i) first to figure out repetution is happening

(ii) How many unique ans required to be saved.

(iii) what planing to save at 'i' th position

(iv) check if ans. is already exits.

0 1 1 2 3 5 8 13 21 34

→ yes ⇒ return

→ no

→ calculate

→ save for future use

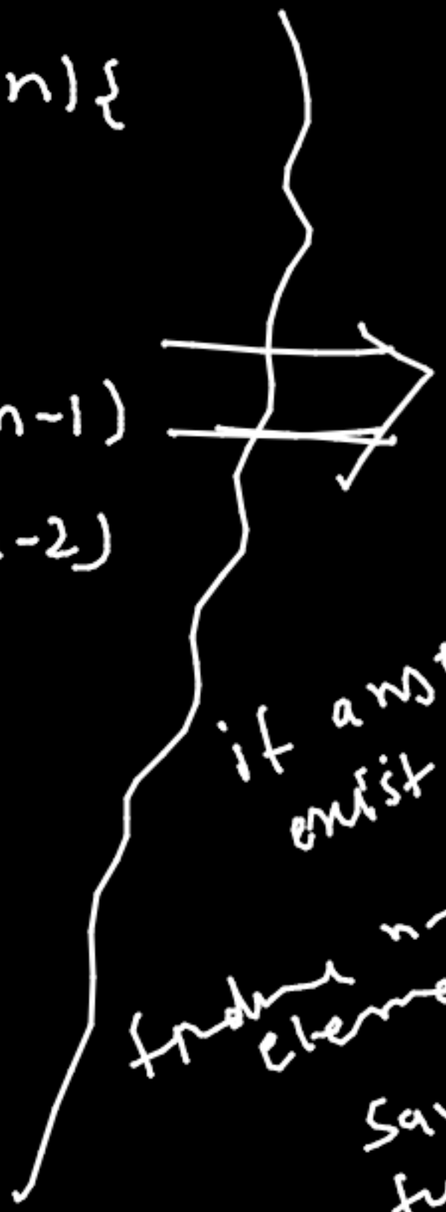
→ return;

ans[i] = ith fibo no.

DP In short

Recursion

```
int fibo (int n){
    if (n <= 1)
        return n;
    int a = fibo (n-1)
    int b = fibo (n-2)
    return a+b;
}
```



if ans[n] already exist return it

```
if (ans[n] != 0)
    return ans[n];
```

find n-1, n-2 element

```
int a = fibo(n-1);
int b = fibo(n-2);
```

saving for future use

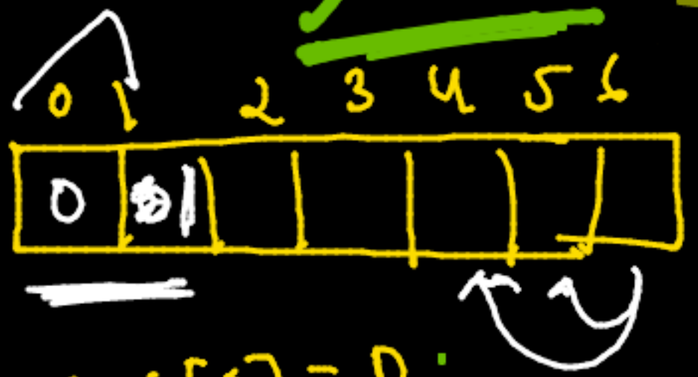
```
ans[n] = a+b;
return ans[n];
```

returning ans

```
int ans[n]; - {0,0,0,0,0,0}
int fibo( int n ) {
    if (n <= 1)
        return n;
    if (ans[n] != 0)
        return ans[n];
    int a = fibo(n-1);
    int b = fibo(n-2);
    ans[n] = a+b;
    return ans[n];
```

Memoization

Smallest DP (Iterative)

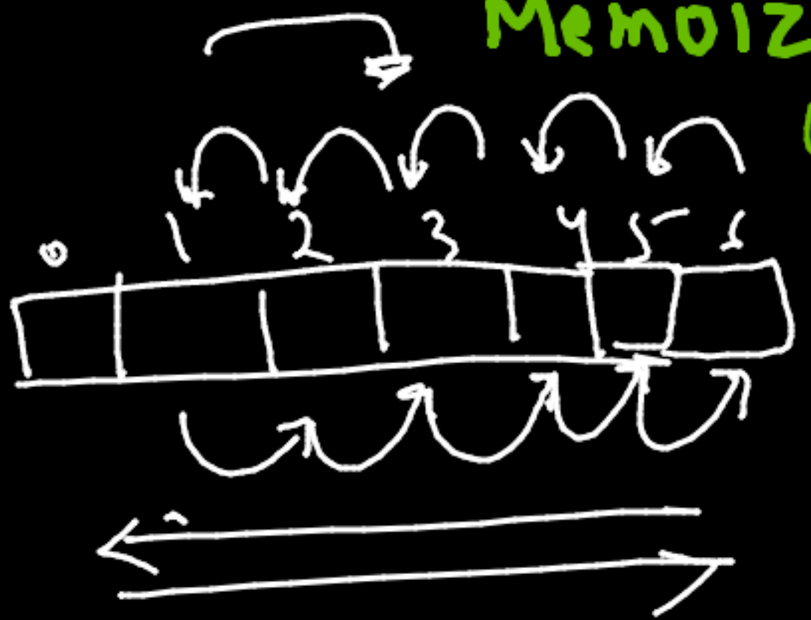



```
ans[0] = 0;
ans[1] = 1;

for (int i=2; i<=n; i++)
    ans[i] = ans[i-1] + ans[i-2];
```

Bottom Up Approach

Memoization (Recursive)



Top Down Approach

Min steps to 1

↳ min steps.

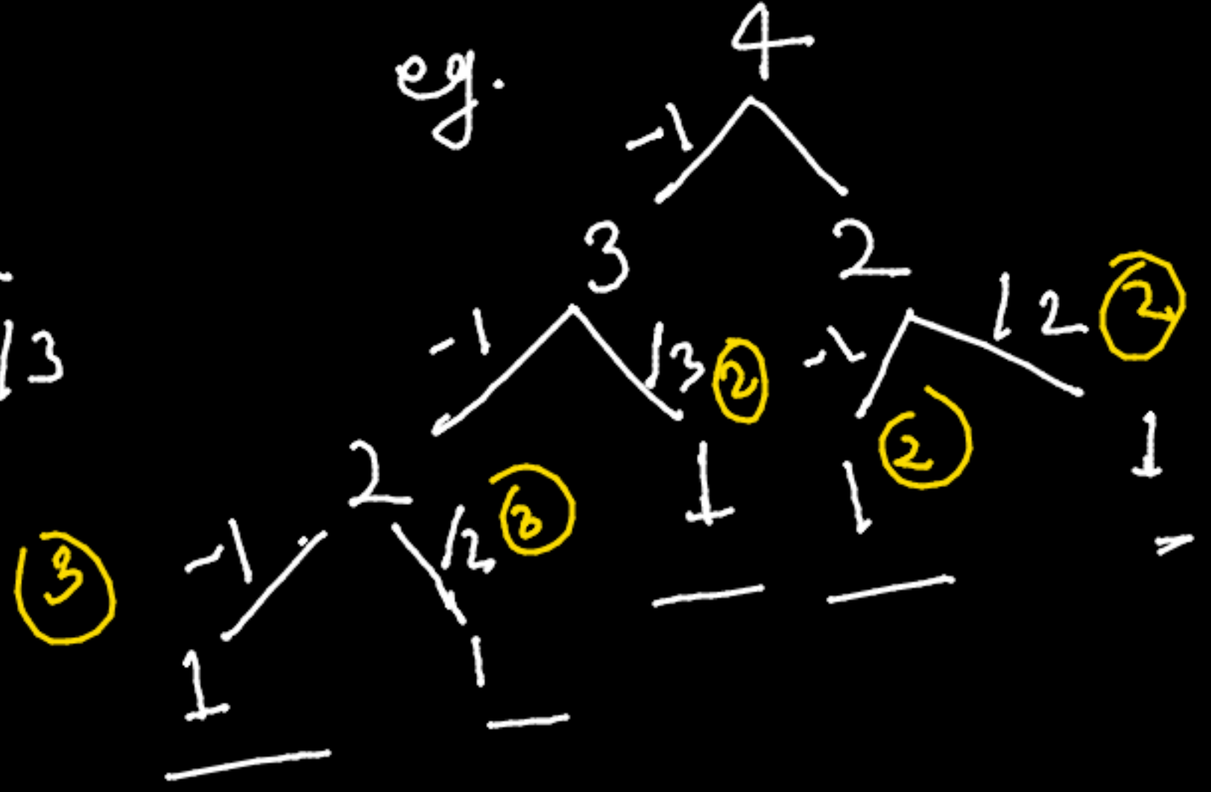
Condition

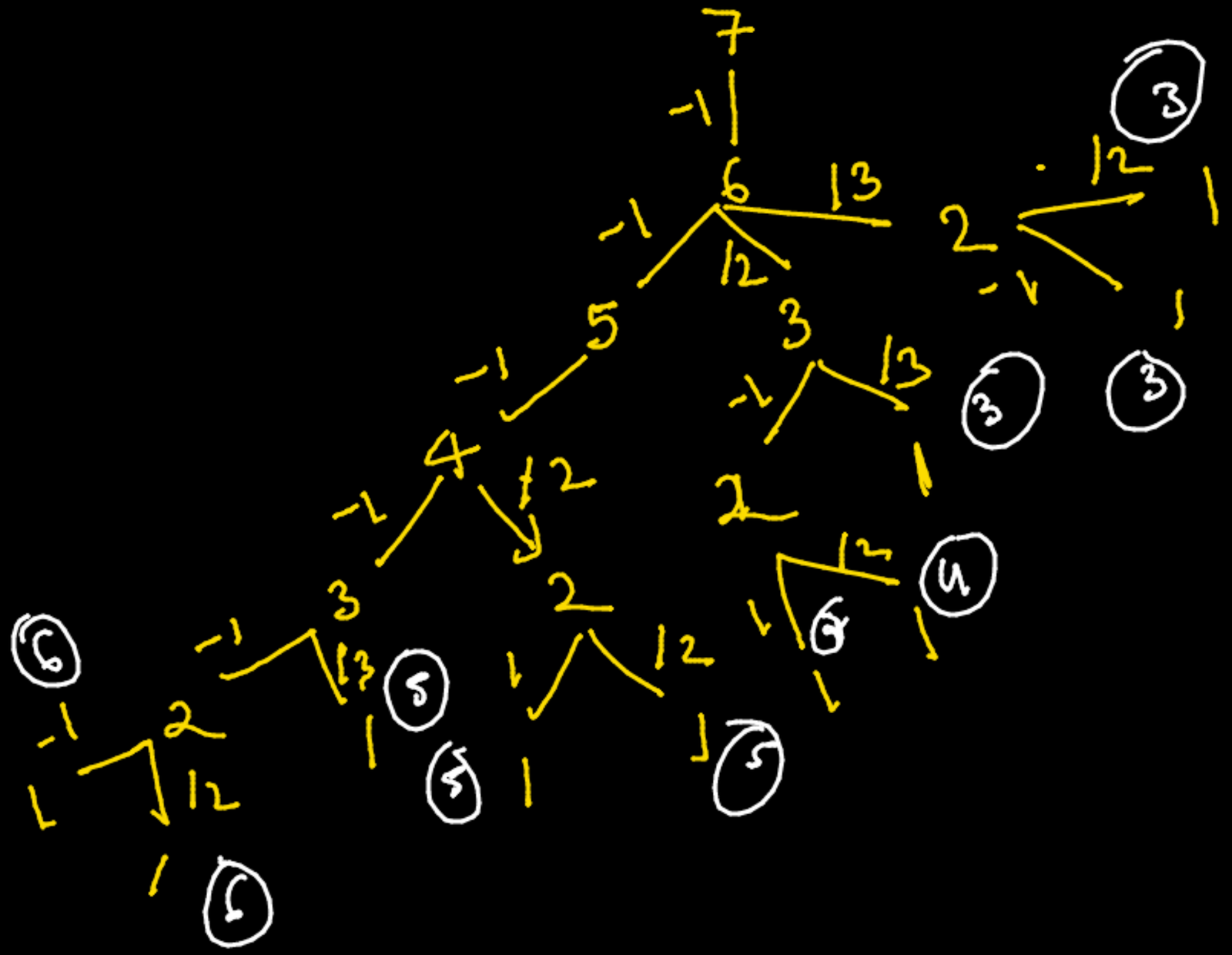
(i) subtract 1

(ii) if n%2 then n = n/2

(iii) if n%3 then n = n/3

eg.

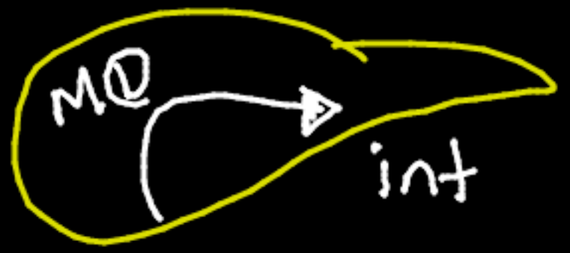




Brutes Force

M → Memorization.

```
int countAns(int n){
    if (n<=1)
        return 0;
    int x = countAns(n-1);
    int y = INT_MAX, INT_MAX;
    if (n%2 == 0)
        y = countAns(n/2);
    if (y%3 == 0)
        z = countAns(n/3);
    return min(x, min(y,z)) + 1;
}
```



Approach

DP.

7 n=7

↳ ans must be 3.

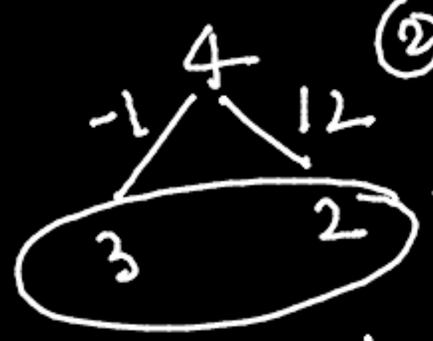
0	1	2	3	4	5	6	7	8
0	0	1	1	2	3	2	3	

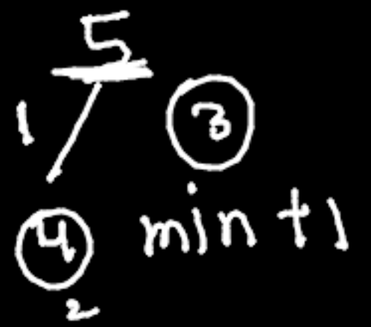
0 → 0

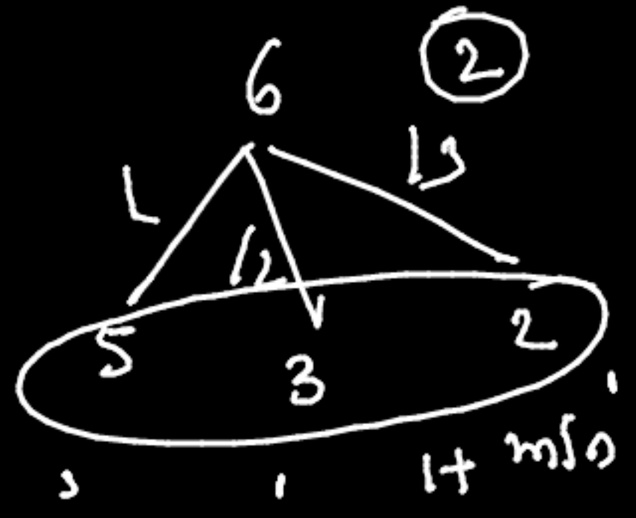
1 → 0

2 → 1

3 → 1







Code

```
fun( int n ) {
    ans[0] = 0
    ans[1] = 0
    ans[2] = 1
    ans[3] = 1

    for (int i = 4; i <= n; i++) {
        int x = ans[ i-1];
        int y = INT_MAX, z = INT_MAX;
        if ( i % 2 == 0)
            y = ans[ i/2 ];
        if ( i % 3 == 0)
            z = ans[i/3];
        ans[i] = 1 + min(x, min(y,z));
    }

    return ans[n];
}
```

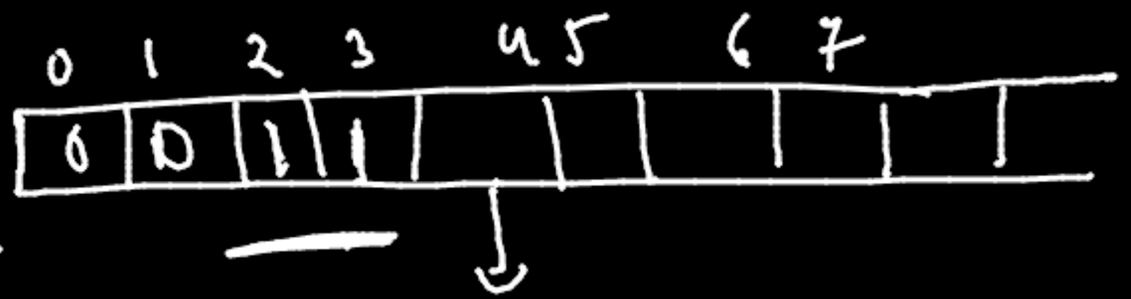
Storing smaller ans

→ Iteration

DP

→ Returning

Memoization




```
//checking ↪ exist or not
if (arr[n] != -1)
    return arr[n];
```

x = (3, arr)
y = (2, arr)

All same as brute force;

```
arr[1] = 1 + min(x, min(y, z));
```

Staircase

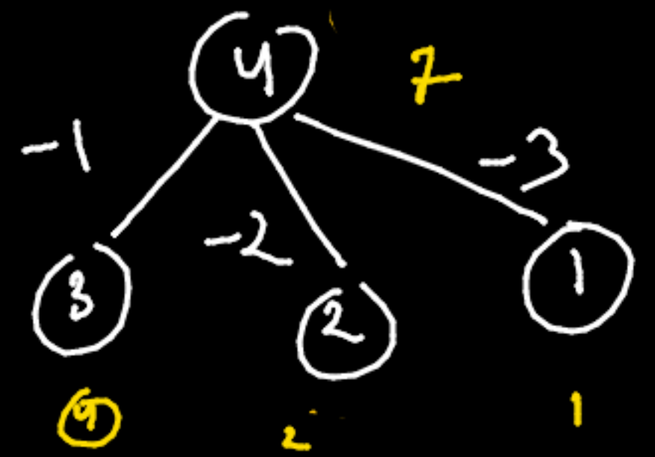
1 or 2 or 3 step at one time

n = 2

1 → 1 (1)

2 → (1,1)
(2)

3 → (1,1,1)
(2,1)
(3)
(1,2)



Brute Force

```
int mod → 10^9 + 7;
int fun( int n) {
  if (n <= 2)
     return n;
  int x = fun(n-1);
  int y = fun(n-2);
  int z = fun(n-3);
  return (x+y+z) % mod;
}
```

1 or 2 or 3

Memoization.

↕ Top down

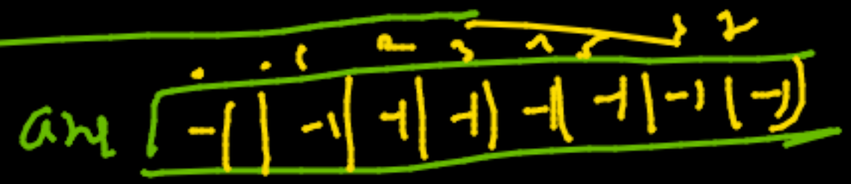
1 → 1 (1)

2 → 2 (1,1) (2)

3 → 4 (1,1,1) (2,1) (1,2) (3)

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

```
ans[n] = x+y+z;
```



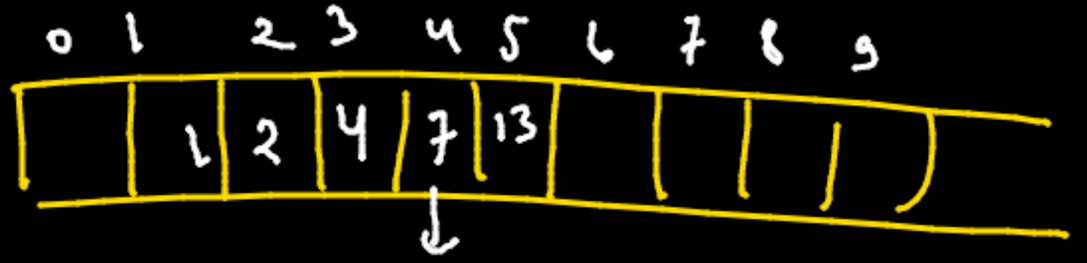
DP:

```
int fun( int n){
    if (n <= 2)
        return n;

    ans[1] = 1
    ans[2] = 2
    ans[3] = 4;

    for (int i=4; i <= n; i++){
        ans[i] = (ans[i-1] + ans[i-2] + ans[i-3])% mod;
    }
    return ans[n];
}
```

Bottom Up



$f(4) = f(3) + f(2) + f(1)$

$f(5) = f(4) + f(3) + f(2)$

$f(4)$ → 7, $f(3)$ → 4, $f(2)$ → 2

Minimum count (of square)

sum of square

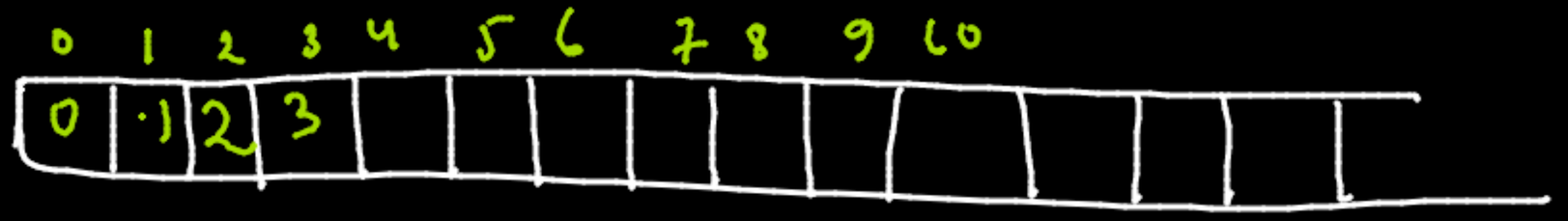
$4 \rightarrow 1^2 + 1^2 + 1^2 + 1^2$ (4)

$4 \rightarrow 2^2$ (1)

$12 \rightarrow 1^2 + 1^2 + 1^2 + 3^2$ (4)

$\rightarrow 2^2 + 2^2 + 2^2$ (3)

n=12




```
if(n <= 3)
    return n;
```

ans[i] ⇒ min no. required to reach at i

ans[0] = 0

ans[1] = 1 — 1^2

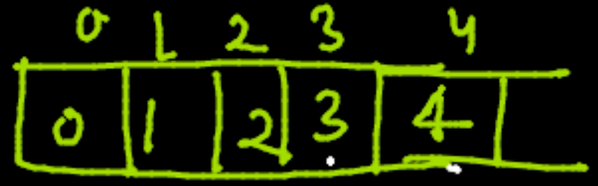
ans[2] = 2 — 1^2+1^2

ans[3] = 3 — $1^2+1^2+1^2$

~~ans[4] = 1~~ — 2^2

DP

```
for (int i=4; i<=n; i++){
    ans[i] = i;
    for (int x=1; x<= ceil(sqrt(i)); x++)
    {
        int temp = x * x;
        if (temp > i)
            break;
        else
            ans[i] = min (ans[i], ans[i-temp]);
    }
}
return ans[n];
```



i = 4

x = 2

temp = 4

ans[4], ans[4], ans[3]; ans[0]
ans[4], ans[0]

Brute Force

```
int minCount(int n) {
    if(sqrt(n) - floor(sqrt(n)) == 0)
        return 1;

    if (n <= 3)
        return n;

    int res = n;

    for (int i=1; i <= sqrt(n); i++) {
        int temp = i * i;
        if (temp > n)
            break;
        else
            res = min(res, 1 + minCount(n-temp));
    }
    return res;
}
```

100

10 - 10 == 0 (✓)

102 10.1.. - 10 == 0 ✗

$1 = 1^2$ ①

$2 = 1^2 + 1^2$ ②

$3 = 1^2 + 1^2 + 1^2$ ③

No. of Balanced BT's

(The absolute diff of height of left subtree and right subtree at any node is 1)

h → 1

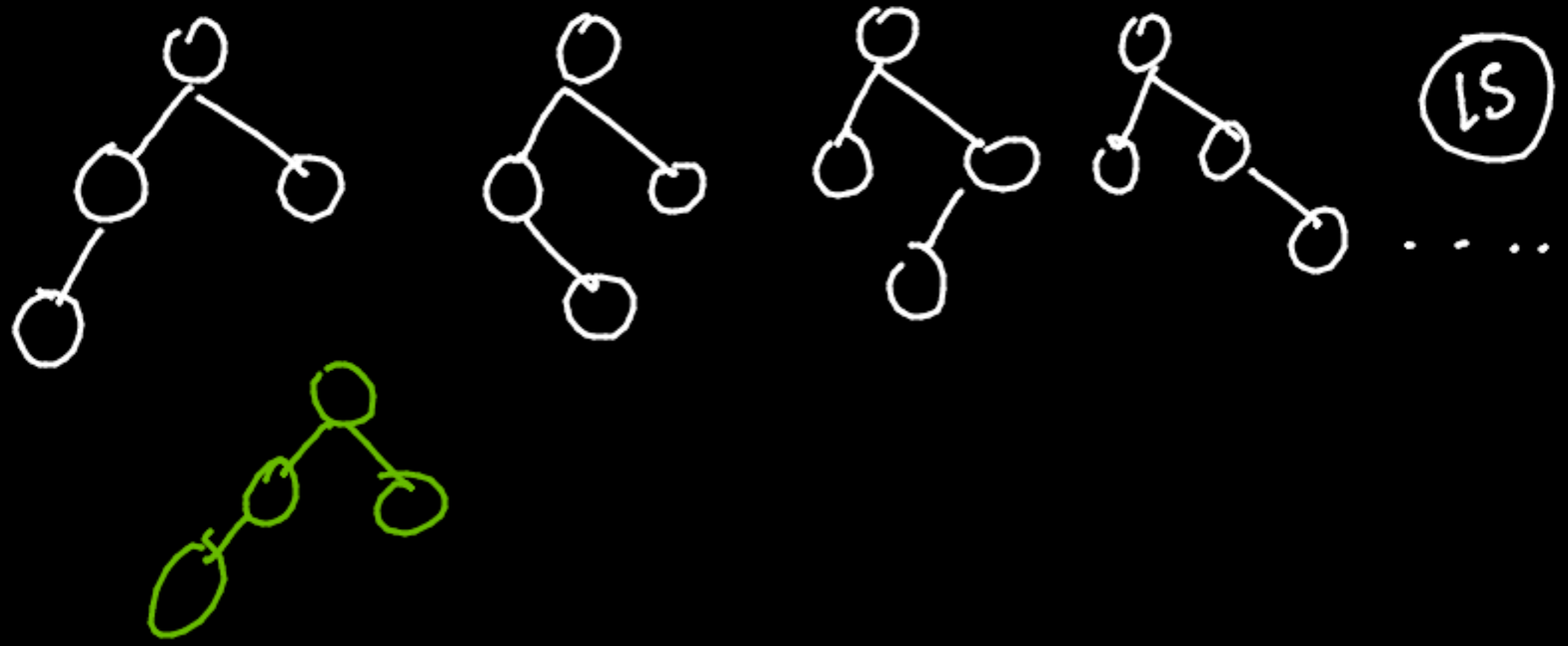
(1)

h → 2

(3)

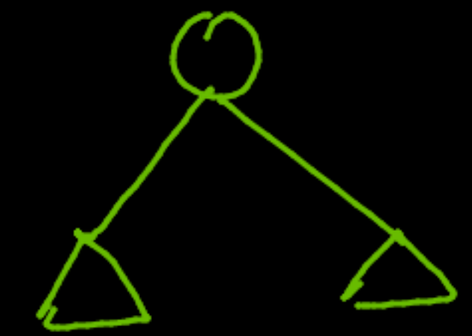
h → 3

(15)



(15)

. . . .




```
int bal(int h){
    if(n<=1)
        return 1;
    int x= bal(n-1) * bal(n-1)
    int y= ------------ * bal(n-2)
    int z; bal(n-2) * bal(n-1)

    return x+y+2;
}
```