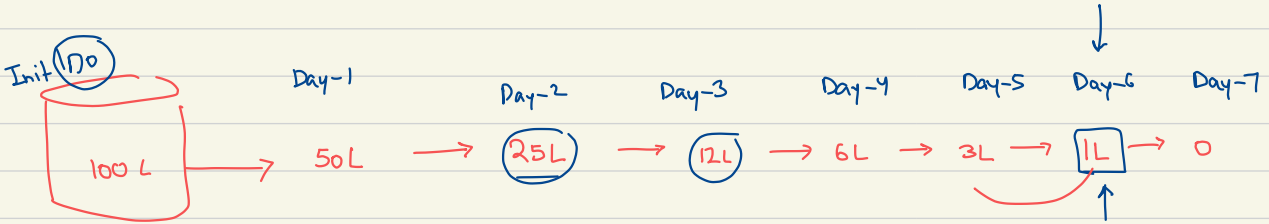
Time Complexity -2

- ↳ Intro to Recursive Functions
- ↳ Time complexity
- ↳ Space complexity

$\log_B N$ Computer Science $B = 2$



100

Day 1 $\frac{100}{2}$ $\frac{100}{2^k} = 1$

Day 2 $\frac{100}{2^2}$ $\frac{100}{4} = 25$ $\Rightarrow$ $100 = 2^k$

Day 3 $\frac{100}{2^3}$ $\frac{100}{8} = 12L$ $\Rightarrow$ $\log_2 100 = \log_2 2^k$

Day k $\frac{100}{2^k} = 1L$ $\Rightarrow$ $k \log_2 2 = \log_2 100$

$\log_2 2 = 1$

$\Rightarrow \quad k = \log_2 100$

$= 6.xx + 1$

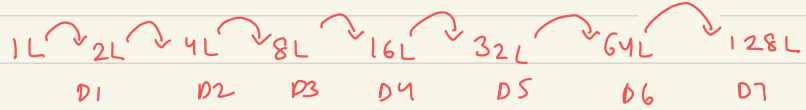
$= 7$ days

$\log_2 16 = \boxed{4}$ ← $2^{\square} = 16$

$\log_B N = \text{Power}$ ← $B^{\text{Power}} = N$

$\log_{10} 1000 = 3$

$$\log_B X^Y = Y \log_B X$$



$\log_2 100 = 7$ 6.xx

⇝ 7 days

use of Log

- `for(i=N ; i>=1 ; i=i/2)` — 1 → 2 → 4 → 8 → 16 → 32 → 64 ; $N \to N/2 \to N/4 \to \cdots \to 1$ — $\log N$
- `for(i=1; i<=N ; i=i*2)` — $1 \to 2 \to 4 \to 8 \to \cdots \to N/2 \to N$ — $\log N$

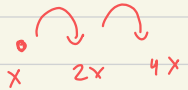
`for(i=1; i<=logN; i++)` — 1, 2, 3 ---- 7 — $\log N$

3

$N = 100$ `for(i=0; i<= N/2 ; i++)`

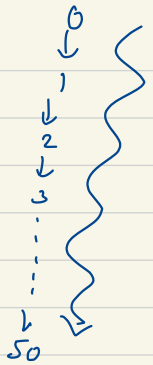
$= \frac{N}{2}$ steps 50 steps

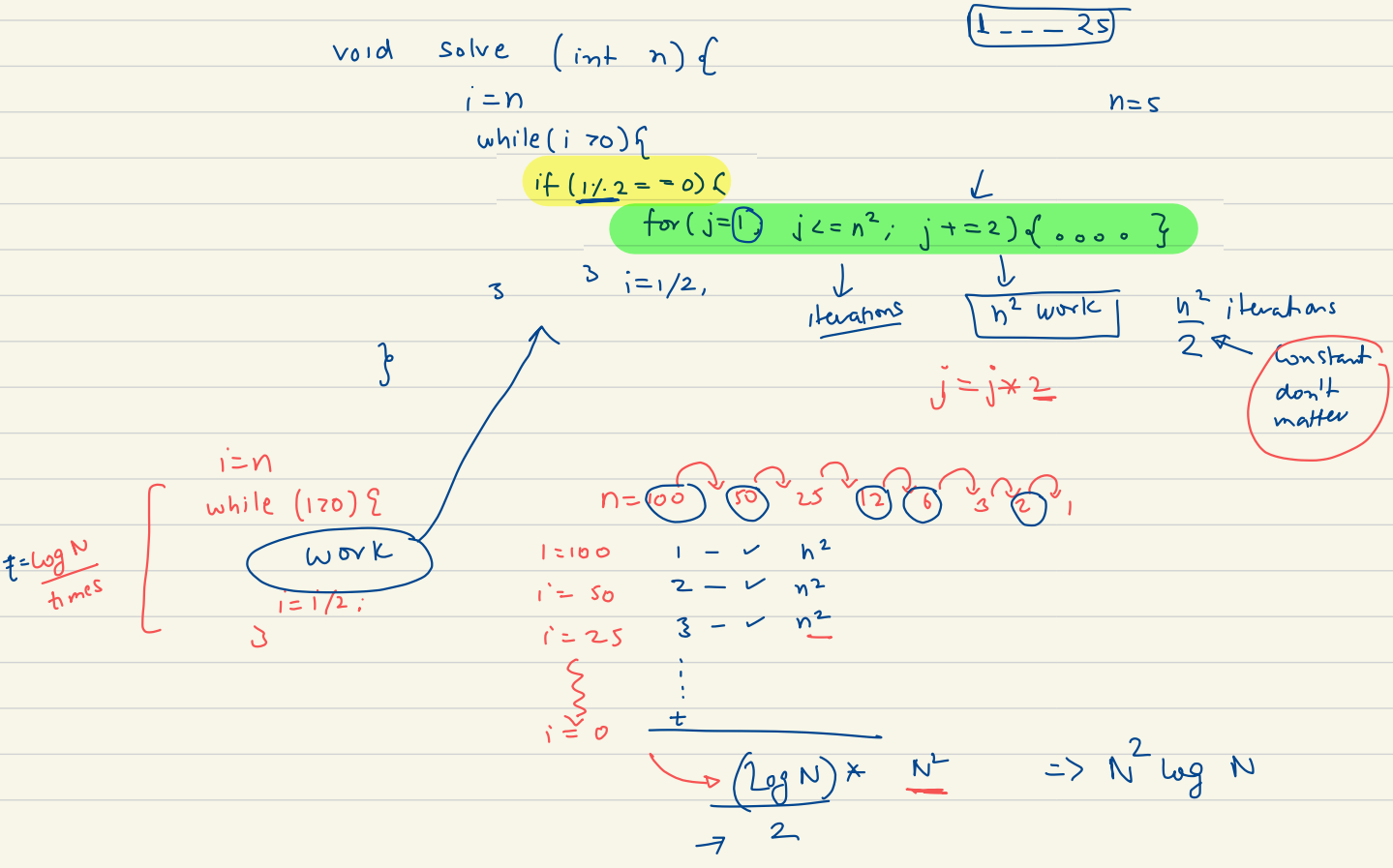
$= O(N)$



$\log_2 50$

$\approx$ 6 steps




1 - - - 25
void solve (int n) {
i = n
n = 5
while (i > 0) {
if (i % 2 == 0) {
for (j = 1; j <= n^2; j += 2) { . . . . }
i = i/2,
iterations
n^2 work
n^2 iterations
2
constant don't matter
j = j * 2
}
i = n
while (i > 0) {
work
i = i/2;
f = Log N
times
n = 100 50 25 12 6 3 2 1
i = 100 1 - ✓ n^2
i = 50 2 - ✓ n^2
i = 25 3 - ✓ n^2
i = 0
+
(log N) * N^2
2
=> N^2 log N

Q

```
for (j=1, j<=N^2 ; j=j*2) {
// work
```

R

1000 ——— 10,00,000

$= \log_2 R^2$ (log R)

$= \log_2 N^2$

$= 2 \log_2 N$

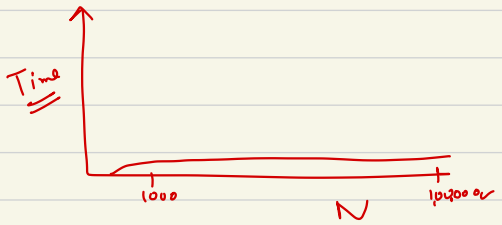
$= O(\log N)$ mean $\hookrightarrow c \log N$

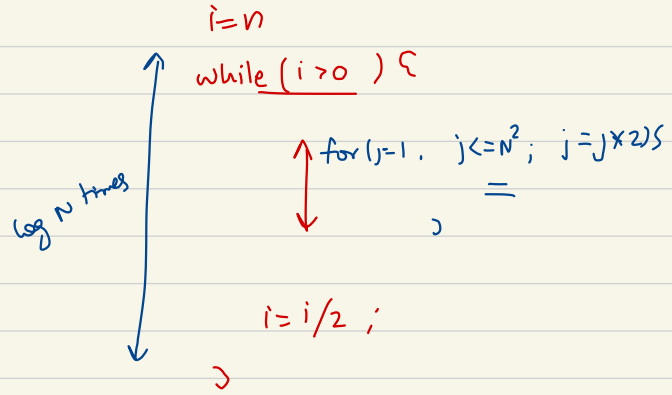
c ↑ constant

N = 1000

$\log_2 1000 \approx 10$

$\log_2 N^2 - \log_2 (1000)^2 = 20$





$(\log_2 N) * (\log_2 N)$

$(\log_2 N)^2$

Not same $\log_2 N^2 = 2 \log N$

$\log 16 = 4$

$(\log 16)^2 = 4 \times 4 = 16$

$\log_2 16^2 = 2 \log 16 = 8$

$\log AB = \log A + \log B$ ✓

$\log 100 \times 100$

$\log_2 25 = \log 5 + \log 5$

$4 = 2 + 2$ ✓

```
for(int i =0;i<arr1.Length; i++)
{
for (int j = 0; j < arr2.Length; j++)
{
if (arr1[i] > arr2[j])
{
Console.WriteLine(arr1[i] + " , " + arr2[j]);
}
```

Nested Loops

Worst case

Constant work

L1 | | | | |

L2 | | | | | |

$O(L1 \cdot L2)$

$O(N \cdot N)$

$O(N^2)$

3

3

i = 0 N

j = 1 N

⋮

N-1

9.45

for(l1){

}

for(l2){

}

l1+l2

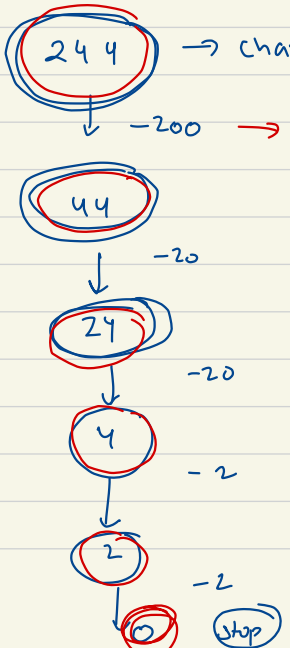
Recursive Functions

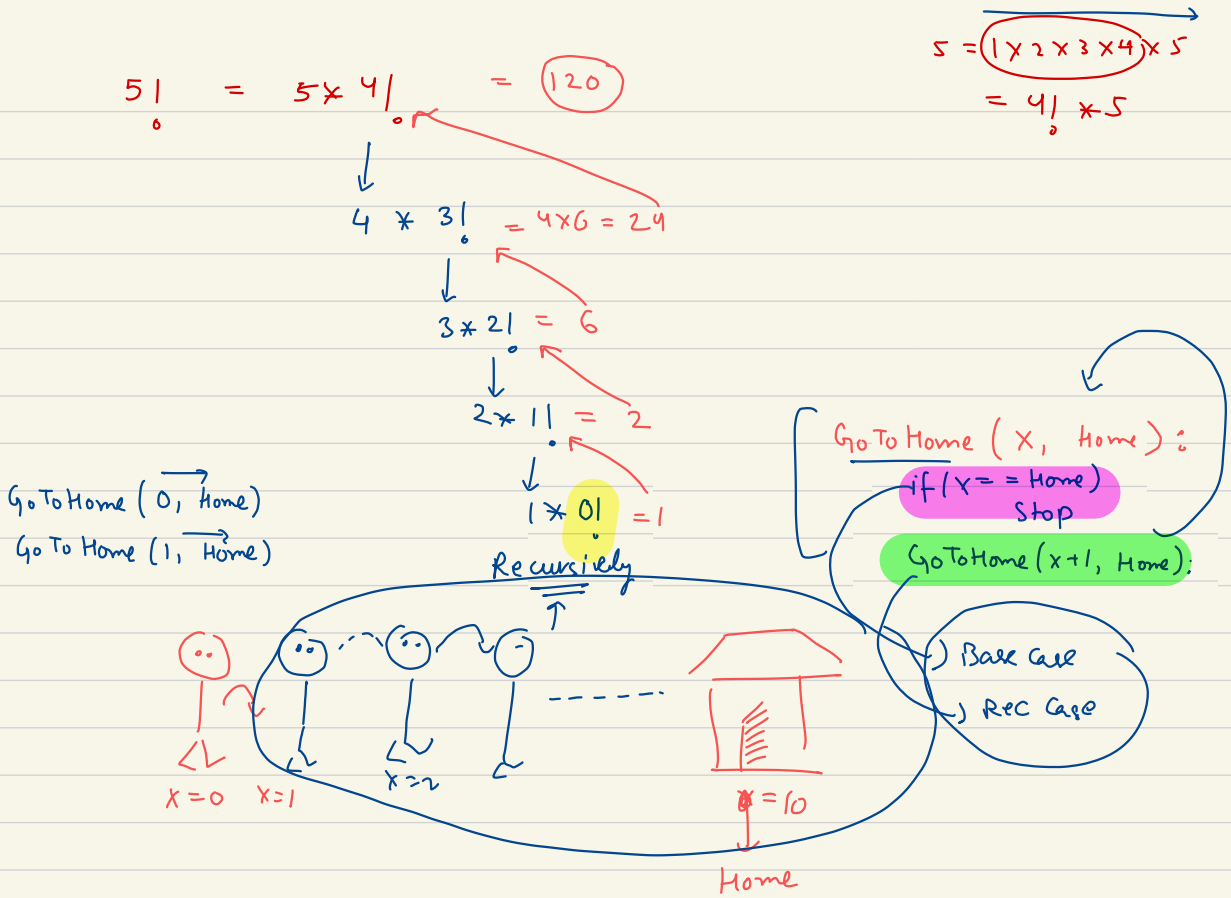
Recursion

indian Coins = [1, 2, 5, 10, 20, 50, 100, 200, 500, 2000]

Given a Complex Problem, break it down into Simple problems of the same type.

loop ✓
Recursion ✓




5! = 5 * 4! = 120
4 * 3! = 4x6 = 24
3 * 2! = 6
2 * 1! = 2
1 * 0! = 1
5 = (1 x 2 x 3 x 4) x 5
= 4! * 5
GoToHome (0, Home)
GoToHome (1, Home)
Recursively
GoToHome (X, Home) :
if (X == Home) Stop
GoToHome (X+1, Home);
Base Case
Rec Case
X=0
X=1
X=2
X=10
Home

PMI

TRICK

Confusing

1. Find out Smallest case.

2. Express big problem in terms of small problem
   ↓
   already solved

Big q
↓
f(n)
↑
Solve

Small problem
↓
K < N
f(k)
↑
Assume it exist

$n!$ $\quad n == 0 \rightarrow 1$

$n!$ $\rightarrow n * (n-1)!$

Space Complexity

↓

Extra space "memory" apart from the input that use to solve any problem.

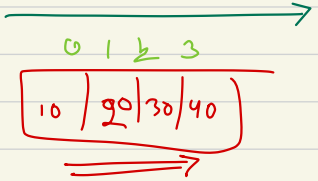
- 3 variables → Constant space → O(1)
- Extra array → that depends upon N.
- Extra stack space → that depends on N

3 variables

```
sum = 0
for (i=1 ; i<=N; i++)
    int no = sc.nextInt();
    sum = sum + no;
```

↳ O(N)

Extra space $\Rightarrow O(N)$



we don't need Storage.

✓ Sum of N Numbers

```
for (i=1; i<= N; i++) {
    arr[i]= sc.nextInt()
}
sum = 0
for (i=0, i<= arr length; i++)
    sum = sum + arr(i);
```

Time & Space in Rec. Code

Time = Total no of Fn calls * Time spent in each call

N * K

= O(N)

Space = Max Space at any point = O(N)

= Max stack frames x Space of each frame

N

K