Recursion – 2

- solving a problem using subproblems

smaller instances of the same problem.

Today.

3 Steps

1. Assumption
2. Main logic
3. Base Condition.

1. P1
2. P2
3. Reccurrence Relations

Q1 Sum of digits

Given a number, calc. its sum of digits.

$N = 124$

sum of digits $= 4 + 2 + 1 = \underline{7}$.

$124 \% 10 = \underline{4}$

$N = 77 \Rightarrow 7 + 7 = 14$

$77 \% 10 = \underline{7}$

$N = 12999 \Rightarrow 1 + 2 + 9 + 9 + 9 = \underline{30}$

$12999 \% 10 = \underline{9}$

```
def sum_of_digits(N):
    if N == 0:
        return 0
```

```
if N<10:
    return N.
```

$a_1 a_2 a_3 --- a_y$

a_y + sum of digits($a_1 --- a_{y-1}$)

```
return (N % 10) + sum_of_digits(N//10)
```

Todo: Write flow

1. Assumption

sum of digits of number N

$N \geq 0$

2. Main Logic

124 % 10 = 4

//10

12

124//10 = 12

12 % 10 = 2

//10

1 % 10 = 1

1//10 = 0

Q2 Implement your own power fn

Given a & N, calculate a^N

Use Recursion

× Can't use **

a**n.

× Can't use Math.pow

E.g. $3^3 = 3*3*3 = 27$

$3^0 = 1$

$3^1 = 3$

$3^2 = 9$

$3^4 = 81 = 3*3^3$

$$\boxed{3^0 = 1}$$

```
def recursive_power(a,N):
    # Assumption: Compute a^N, N>=0

    # Base Case
    if (N==0):
        return 1
    # Main Logic

    return a * recursive_power(a, n-1)
```

$a^N = a \cdot a^{N-1}$

$3^2 = 3 \cdot 3^1$

$3^3 = 3 \cdot 3^2$

$3^4 = 3 \cdot 3^3$

```
n==1:
    return a
```

$3+3+3$

$3 \cdot 3 \cdot 3$

Flow for $a^{\boxed{8}}$

$\downarrow$

$a * a^7$

$\downarrow$

$a * a^6$

$\downarrow$

$a \cdot a^5$

$\downarrow$

$a \cdot a^4 \longrightarrow a \cdot a^3 \rightarrow a \cdot a^2 \rightarrow a \cdot a^1 \rightarrow a \cdot \underline{\underline{a^0}}$

N recursive calls.

Time Complexity O(N)

Space Complexity O(N)

Optimization

$a \cdot a \cdot \text{pow}(a, n-2)$

$8 \downarrow 6 \downarrow 4 \downarrow 2 \downarrow 0$

$O(N/2)$

$= O(N)$

a^{16}

$\downarrow$

$a^8 \cdot a^8$

$\downarrow$

$a^4 \cdot a^4$

$\downarrow$

$a^2 \cdot a^2$

$\downarrow$

$a^1 \cdot a^1$

$\downarrow$

$\log_2 16 = 4$

even % 2 == 0

$$a^{15} = a \cdot \underbrace{a^{14}}$$

$$\downarrow$$

$$\underbrace{a^{7}} \cdot a^{7}$$

$$\downarrow$$

$$a \cdot a^{6}$$

$$\downarrow$$

$$a^{3} \cdot a^{3}$$

$$\downarrow$$

$$a \cdot a^{2}$$

$$\downarrow$$

$$a \cdot a$$

$a^{10} = a^5 \cdot a^5$

N is even

$\downarrow$

$\frac{N}{2} + \frac{N}{2}$

even

pull a single a out

$a \cdot a^4$

$\downarrow$

$a^2 \cdot a^2$

$\downarrow$

$a \cdot a$

$\downarrow$

$a \cdot a^0$

$a^1 = a \cdot a^0$

odd $7 - 1 = 6$

$15 - 1 = 14$

$9 - 1 = 8.$

Main logic

Main logic.

$$a^N = \begin{cases} a^{N/2} \cdot a^{N/2} & \text{, if } N \text{ is even} \\ a \cdot a^{N-1} & \text{, if } N \text{ is odd} \end{cases}$$

$7 // 2 = 3$

$$a^7 = a \cdot a^6 = a \cdot a^3 \cdot a^3$$

$(N-1)//2$

$N//2$

$\frac{n-1}{2}$

N	$N-1$	$(N-1)//2$	=	$N//2$
1	0	0	=	0
3	2	1	=	1
5	4	2	=	2
7	6	3	=	3
9	8	4	=	4
11	10	5	=	5
13	12	6	=	6

$(N-1)//2 = N//2$

pow1

def optimized_power_1(a, N):

$N \geq 1$

N=18 → 9

N=20 → 10

\# a^N

\# Base

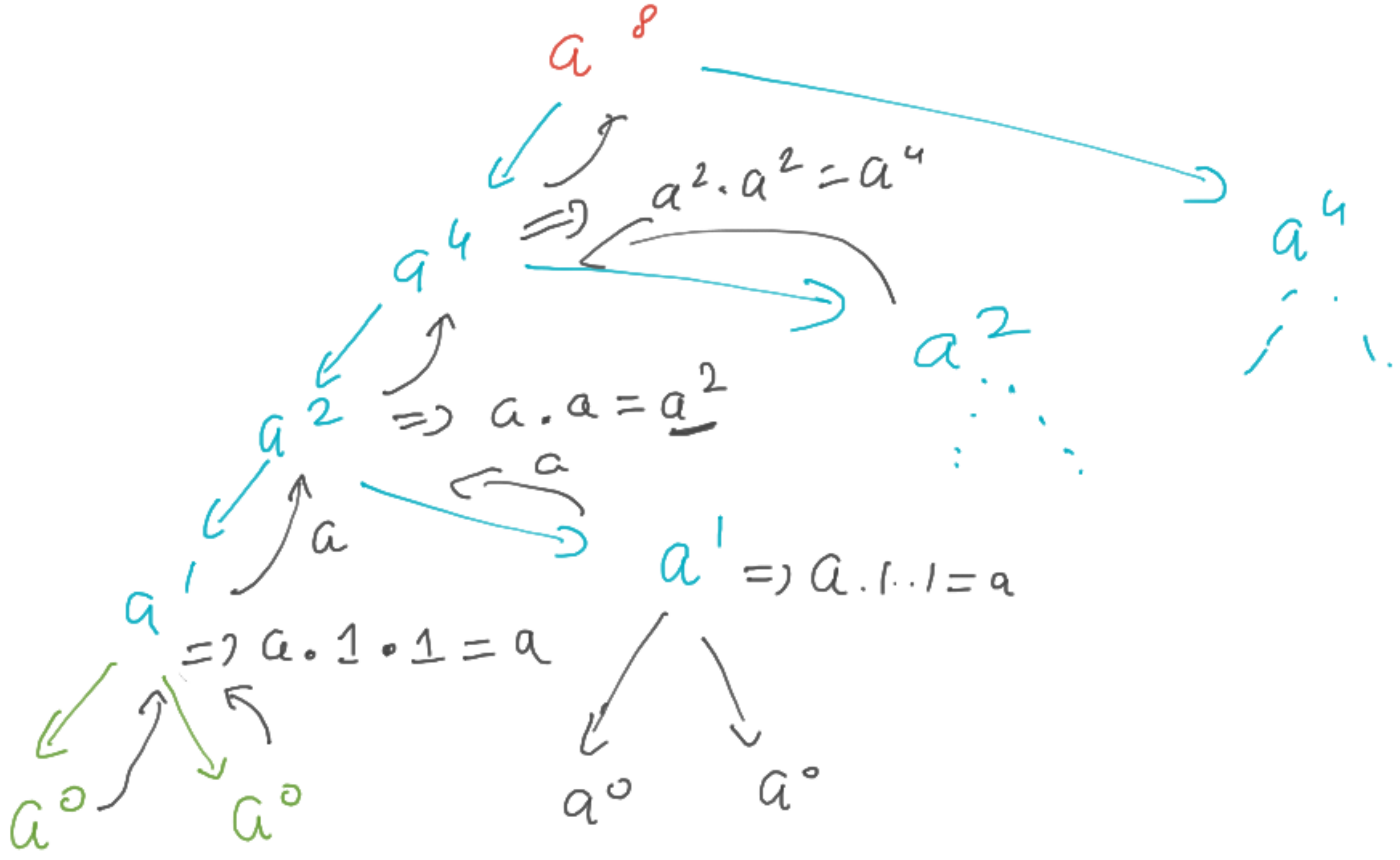
if N==0:
    return 1

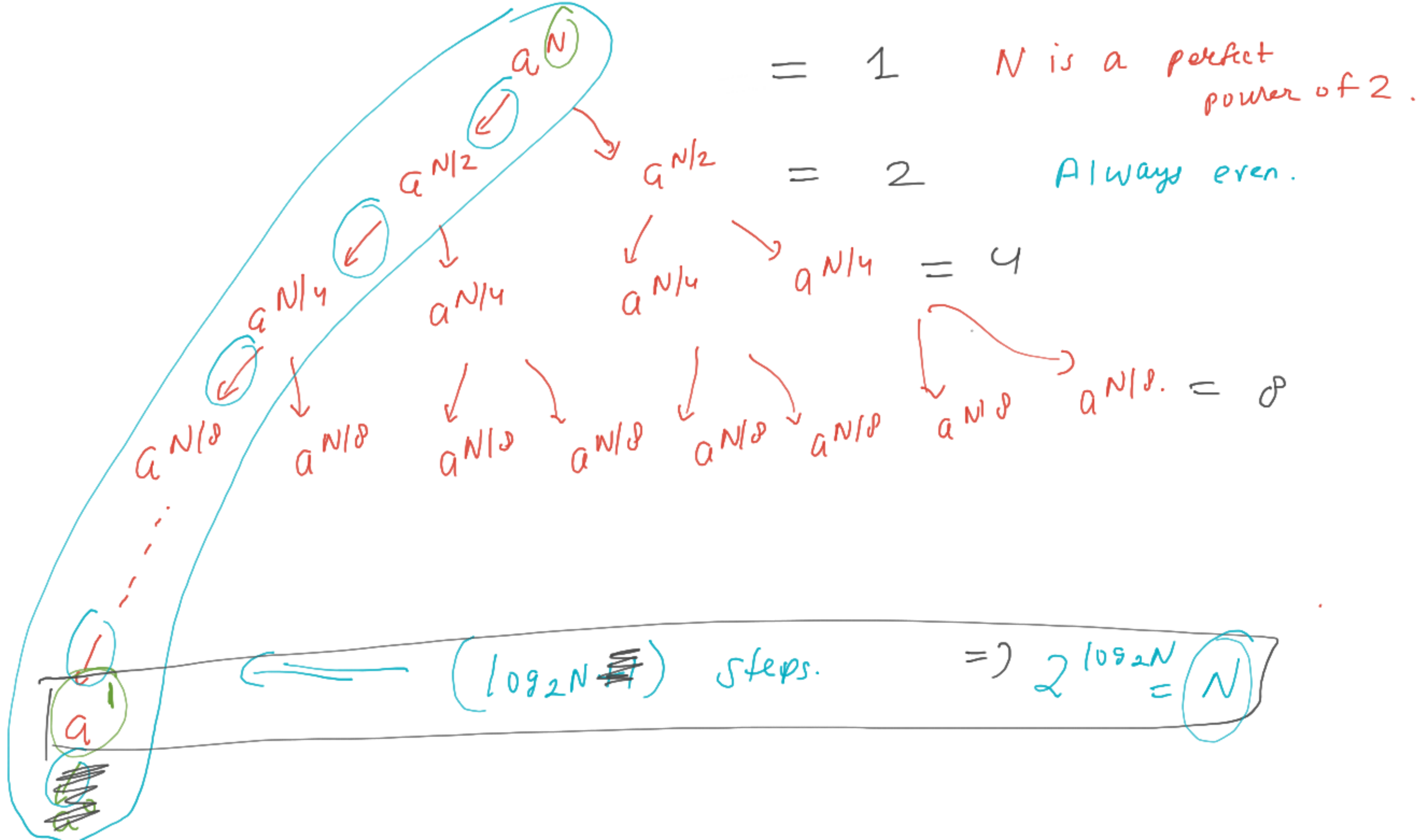
if N==1:
    return a

\# Main logic

if (N & 1 == 0):
    return pow1(a, N//2) * pow1(a, N//2)
else:
    return ~~a * pow1(a, N-1)~~
    → a * pow1(a, N//2) * pow1(a, N//2)

Flow for a^8





$a^N = 1$
N is a perfect power of 2.
$a^{N/2} = 2$
Always even.
$a^{N/4} = 4$
$a^{N/8} = 8$
a^{1}
$(\log_2 N)$ steps.
$\Rightarrow 2^{\log_2 N} = N$

$N = 8$

$\Rightarrow a^8$ $\quad 1$

$\Rightarrow a^4 \quad a^4 \quad \Rightarrow 2$

$\Rightarrow a^2 \quad a^2 \quad a^2 \quad a^2 \quad \Rightarrow 4$

$\Rightarrow a^1 \quad a^1 \quad a^1 \quad a^1 \quad a^1 \quad a^1 \quad a^1 \quad a^1 \quad \Rightarrow 8 = 2^{\log_2 N} = N$

$= 2^3$

How many steps to reach 1

$\Rightarrow \log_2 N$ steps.

$= \log_2 8 = 3$

$8 + \underbrace{4 + 2 + 1}_{7}$

$= 8 + 7$

$= 15 < 2 \cdot 8$
$= 16$

$O(2N)$

$\underbrace{1 + 2 + 4 + 8 + \cdots + N/2}_{\simeq N} + N$

$\simeq 2N$

$$N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \cdots + \infty$$

$$\frac{a}{1-r} = \frac{N}{1-\frac{1}{2}} = \frac{N}{\frac{1}{2}} = 2N.$$

$r = 1/2$

$O(N)$

```
def optimized_power_2(a, N):        N>=1

    if(N==1):
        return a

    halfPower = optimized_power_2(a, N//2)

    if (N&1 == 0):
        return halfPower * halfPower        O(1)
    else
        return a * halfPower * halfPower    O(1)
```

---

optimized_power_2 (3, 8)

Flow for a^8

$a^8 \Rightarrow a^4 * a^4 = \underline{\underline{a^8}}$

a^4

$a^4 \Rightarrow a^2 * a^2 = a^4$

$\underline{\underline{a^2}}$

$a^2 \Rightarrow$ even $\Rightarrow a * a = a^2$

a

a^1

TC: $\underline{\underline{O(\log_2 N)}}$, SC: $O(\log_2 N)$

$\leq 1000 * 1000$ $\quad$ $2*2$

$a*b$

$= O(1)$

$a^9 \Rightarrow$ odd $\Rightarrow \underbrace{a \cdot a^4}_{a^5} \cdot a^4 = a^9$

$a^4 \Rightarrow$ even $= a^2 \cdot a^2 = a^4$

$a^2 \Rightarrow$ even $= a \cdot a = a^2$

a^1 $\quad$ a

Recursion

Recursive Relation

Let us assume $T(N)$ is the time complexity for problem of size N.

Known. $T(0) = O(1)$
$= 1$

```
def sum(N):            -> T(N)
1       if (N==0):     -> O(1)
2           return 0   -> O(1)
3   t = sum(N-1)       -> T(N-1)
4   return (t + N)     -> O(1)
```

$$T(N) = T(N-1) + \underbrace{O(1) + O(1) + O(1)}_{+3}$$

C = constant
$= O(1)$

Recursive Relation for T.C.

$$\underbrace{T(N)}_{\text{unknown}} = \underbrace{T(N-1)}_{\text{unknown}} + O(1)$$

Step-1 : Construct the recursive relation.

Solve?

$$T(N) = \underbrace{T(0)}_{\text{Known}} + \underbrace{\qquad}_{\text{Known}}$$

Substitution method

Step-2 Generalize the expression

so that we can put the base

Goal => all the things on write side as known.

$T(0) = 1$

$T(N) = T(N-1) + 1$

Substitute N with $(N-1)$

$\boxed{T(N-1) = T(N-2) + 1}$

$\Downarrow$

$T(N-2) = T(N-3) + 1$

$O(1) = 1$

$\Rightarrow T(N) = T(N-2) + 1 + 1$

$= T(N-2) + 2$

$= T(N-3) + 2 + 1$

$= T(N-3) + 3$

Some variable K $\Rightarrow$ $\boxed{T(N) = T(N-K) + K}$

Step-3 Put the base condition $T(0) = 1$

Try to find K using base condition

$$T(N) = T(N-K) + K$$

What is the value of K we should chose to have all known things on RHS.

$N - K = 0$

$\Rightarrow \boxed{K = N}$

Step-4 Substitute K val

$$T(N) = T(N-N) + N$$
$$= T(0) + N$$
$$= N + 1 = O(N)$$

Steps

1. Build recursive relation $\boxed{T(N)}$

2. Generalize expression in terms of K

   K = After how many steps we reach the base condition

   = Depth of call stack

   = Space complexity.

3. Put the base condition to find K

4. Substitute the val of ~~K~~ to get $\boxed{T(N)}$.

N

N
$\downarrow$
$N-1$
$\downarrow$
$N-2$
$\downarrow$
$\vdots$
$N-K$
$\vdots$
1
$\downarrow$
0

1 Optimized Power 1

$$T(N) = 2T(N/2) + 1$$

```
def op1(a, n)          n >= 1   =>   T(N)
    if n == 1:
        return a
    if n % 1 == 0
        x = op1(a, n//2)
        y = op1(a, n//2)
        return x * y
    else
        return a * op1(a, n//2) * op1(a, n//2)
```

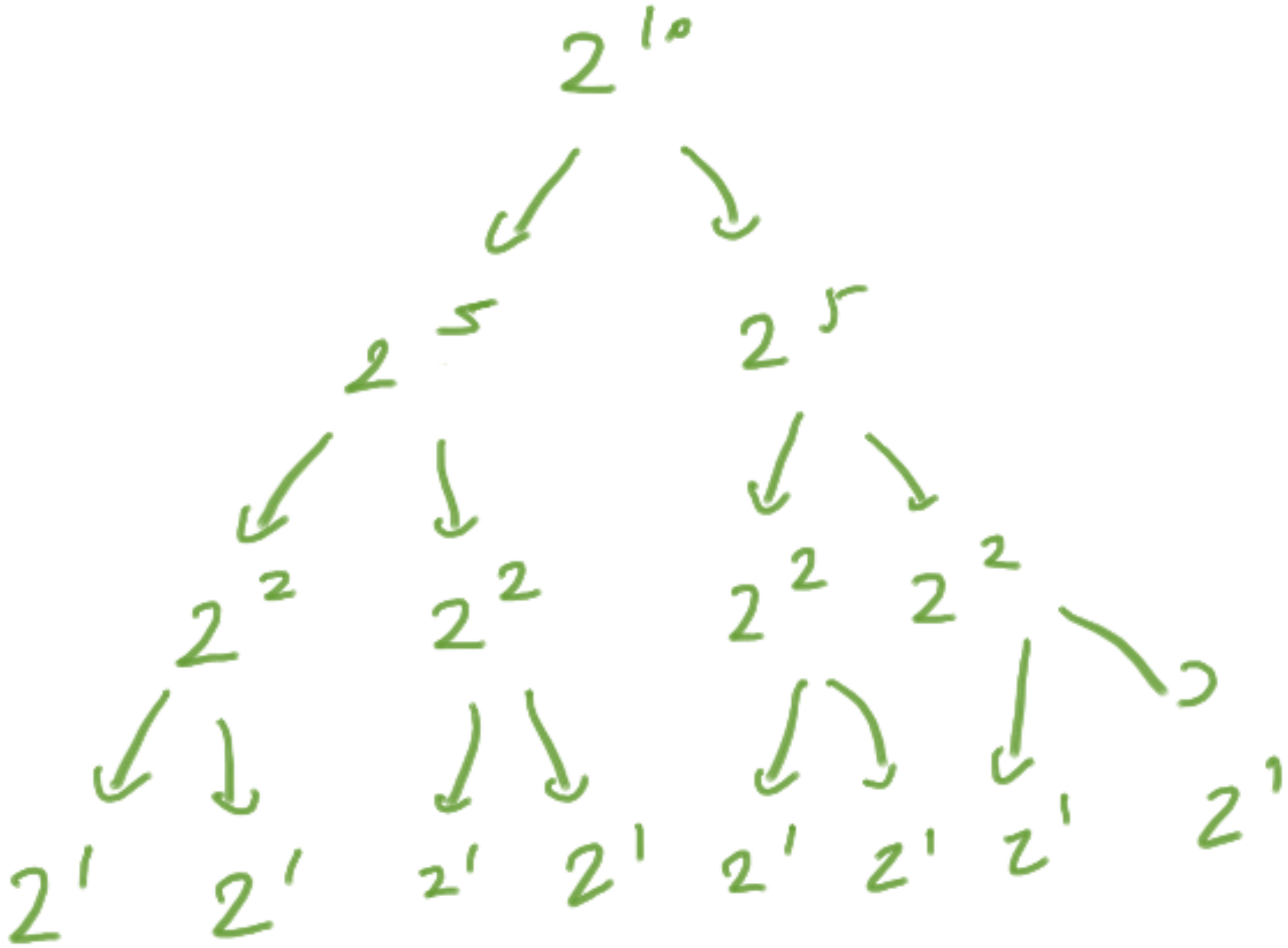
O(1)

T(N/2)   T(N/2)

$$T(1) = 1$$

$$\underbrace{op1(a, n//2)}_{T(N/2)} * \underbrace{op1(a, n//2)}_{T(N/2)}$$

$$2^{10}$$
$$\downarrow$$
$$2^5 \cdot 2^5$$



2

Generalize

(i) $T(N) = 2T(N/2) + 1$

$T(N/2) = (2T(N/4) + 1)$

ii) $T(N) = 2(2T(N/4) + 1) + 1$

$= (2*2)\, T(N/4) + (2+1)$

$T(N/4) = 2T(N/8) + 1$

iii) $T(N) = 2*2*(2T(N/8) + 1)$

$+ (2+1)$

$= 2*2*2\, T(N/8)$

$+ (4+2+1)$

$$\boxed{T(N) = 2^k\, T\left(\frac{N}{2^k}\right) + (2^k - 1)}$$

$1 + 2 + 4 + \dots + 2^{k-1} = (2^k - 1)$

3

$$T(N) = 2^K T\left(\frac{N}{2^K}\right) + (2^K - 1)$$

We know $T(1) = 1$

$$\left(\frac{N}{2^K}\right) = 1 \implies 2^K = N$$

$$\implies \underline{\underline{K = \log_2 N}} \implies \underline{\underline{\text{space}}}$$

4 Substitute value K

$$\implies T(N) = \boxed{2^{(\log_2 N)}}\, T(1) + \left(2^{\log_2 N} - 1\right)$$

$$= (N * 1) + N - 1$$

$$= 2N - 1 \implies O(N) \text{ time.}$$

$T(1) = 1$

```
def op2(a, N):                 => T(N)
    if (N == 1):
        return a               O(1)
    hp = op2(a, N//2)          => T(N/2)
    if N&1 == 0
        return hp*hp           O(1)
    else
        return a*hp*hp
```

$$T(N) = T(N/2) + 1$$

i) $T(N) = T(N/2) + 1$

ii) $T(N) = T(N/4) + 2$

iii) $T(N) = T(N/8) + 3$

$\Downarrow$

Generalize

$$\boxed{T(N) = T\left(\frac{N}{2^K}\right) + K}$$

$T(N/2) = (T(N/4) + 1)$

$T(N/4) = (T(N/8) + 1)$

③ Find K

$$T(N) = T\left(\frac{N}{2^K}\right) + K$$

Known

$$\boxed{T(1) = 1}$$

$$\frac{N}{2^K} = 1$$

$$\Rightarrow K = \log_2 N$$

S.C.

④ Substitute K

$$\Rightarrow T(N) = T(1) + \log_2 N$$

$$= 1 + \log_2 N$$

$$= \boxed{O(\log_2 N)}$$

T.C.

$$\cancel{\frac{N}{2^K} = 0}$$

$$\cancel{\Rightarrow N = 0}$$

HW: $\boxed{T(N) = 2T(N/2) + N}$

$T(1) = 1$

T. C.

$$\boxed{T(N) = T(N-1)+1} \times$$

Master's Theorem

$$T(N) = a \cdot T(N/b) + O(N^c)$$

$a = ?$
$b = ?$
$c = ?$

0. If you can write a recursive relⁿ in this form.

Step-1 Find $t: \log_b a$

Step-2 Compare t with c.

i) $t > c \Rightarrow T(N) = O(N^t)$

ii) $t = c \Rightarrow T(N) = O(N^c \log N)$

iii) $t < c \Rightarrow T(N) = O(N^c)$

(1) $T(N) = 2T(N/2) + 1 \Rightarrow \underline{\underline{N^0}}$

$$\boxed{T(N) = aT(N/b) + O(N^c)}$$

$$\begin{cases} a = 2 \\ b = 2 \\ c = 0 \end{cases}$$

$$t = \log_b a = \log_2 2 = 1$$

(2) $t = 1$ $\quad t > c \Rightarrow O(N^t) = \underline{\underline{O(N^1)}}$

$c = 0$

(2) $T(N) = T(N/2) + 1$ N^0

$T(N) = aT(N/b) + O(N^c)$

$a = 1$

$b = 2$

$c = 0$

$t = \log_b a = \log_2 1 = 0$

$\Rightarrow$ $\underline{\underline{t = c}}$ $\Rightarrow T(N) = O(N^c \log N)$

$= O(N^0 \cdot \log N)$

$= O(\log N)$

③ $\boxed{T(N) = 2T(N/2) + N^1}$ Merge Sort

$a = 2$

$b = 2$

$c = 1$

$t = \log_2 2 = 1$

$t = c \Rightarrow T(N) = O(N^c \log N)$

$= O(N^1 \log N)$

$= \boxed{O(N \log N)}$

⇒ Doubts

bar(x, y) ⇒ x * y.

foo(x, y):

    if y == 0 :

        return 1

    ~~return bar(x, foo(x, y-1))~~

    return x * foo(x, y-1)

$$x^y = x \cdot x^{y-1}$$

OOP Classes, Principles of OOPS.

LL.