↳ Content

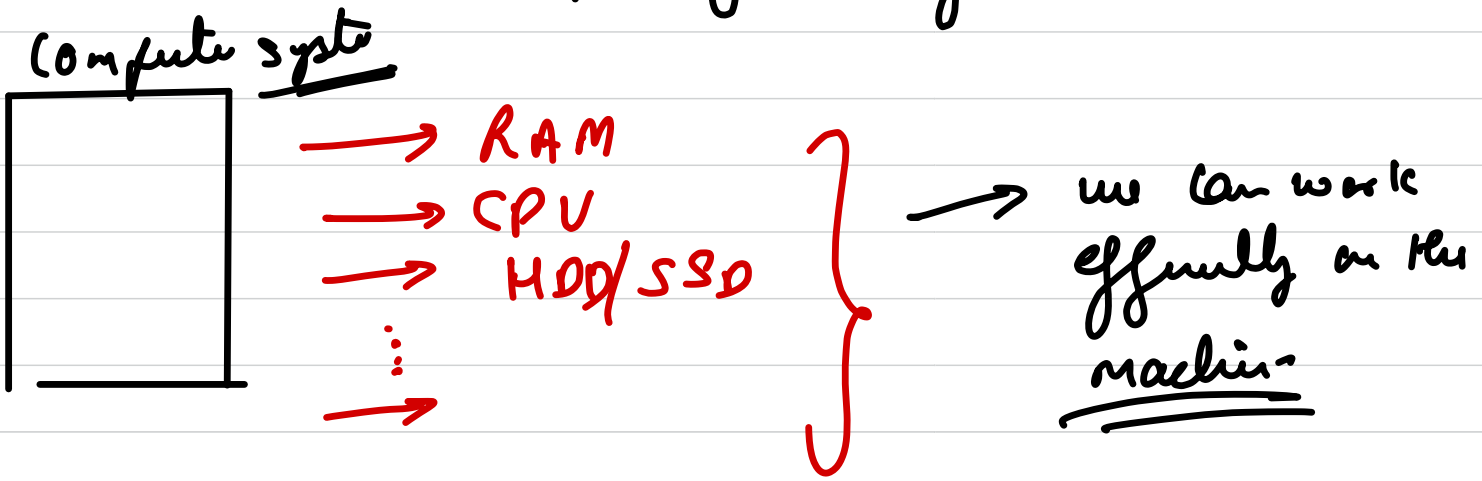
- Recurrence Relation → Complexity analysis
- → Efficiently solve recurrence relation (matrix expo)
- → Bit manipulation → DP with Bitmask
- → Combinatorics (PnC)
- → Probability → DP with probability, Expectation
- → Pigeonhole principle
- →
- →

Pre-requisite → Programming constructs

Recursion

NT

Complexity Analysis

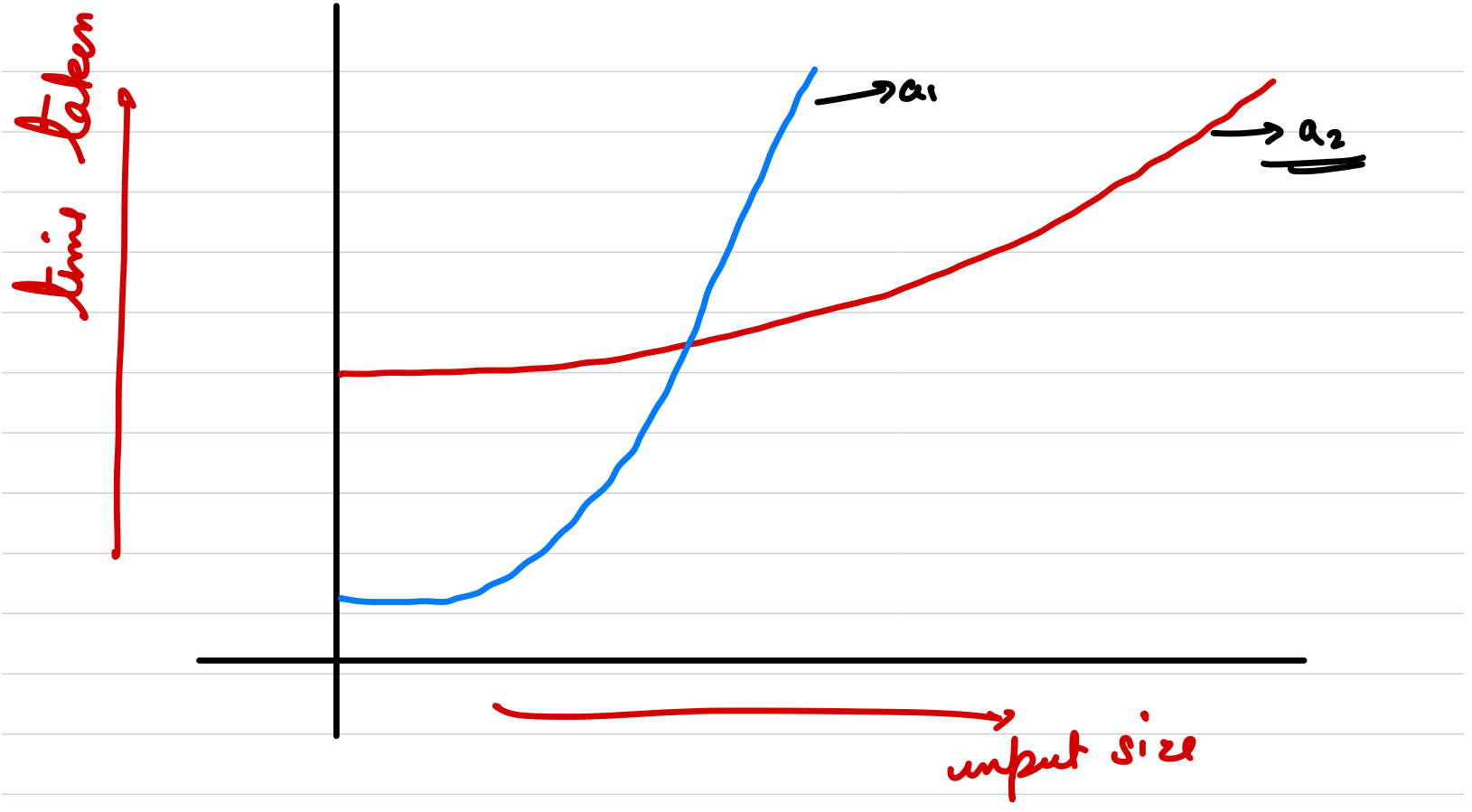


Software → efficient → Time
efficient → Space

1 sec → approx $10^8 - 10^9$

→ **algorithm** → How can we time taken by the algorithm ??

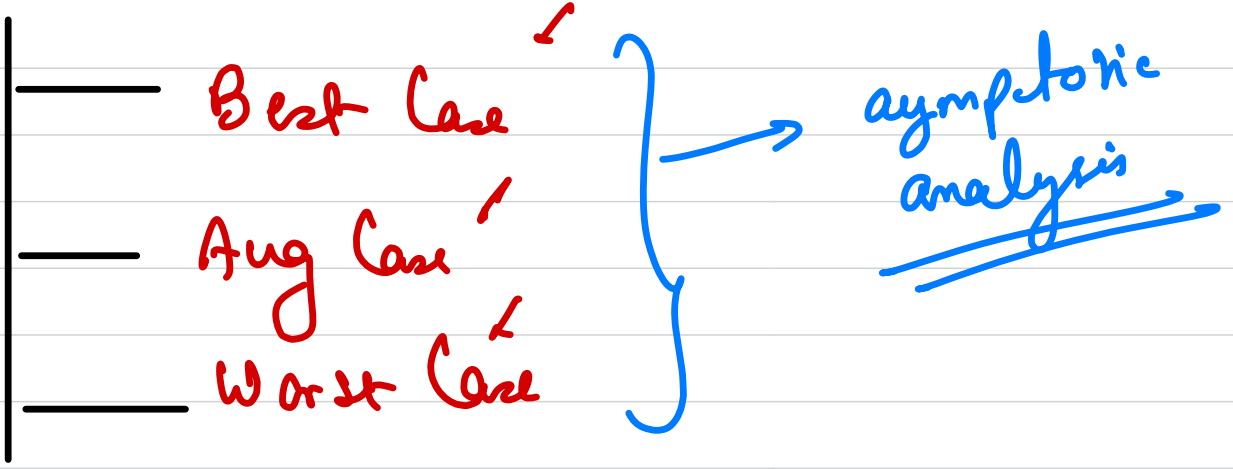
- execution time
- No. of statements executed
- how the algorithm performs on a input and it's growth rate.

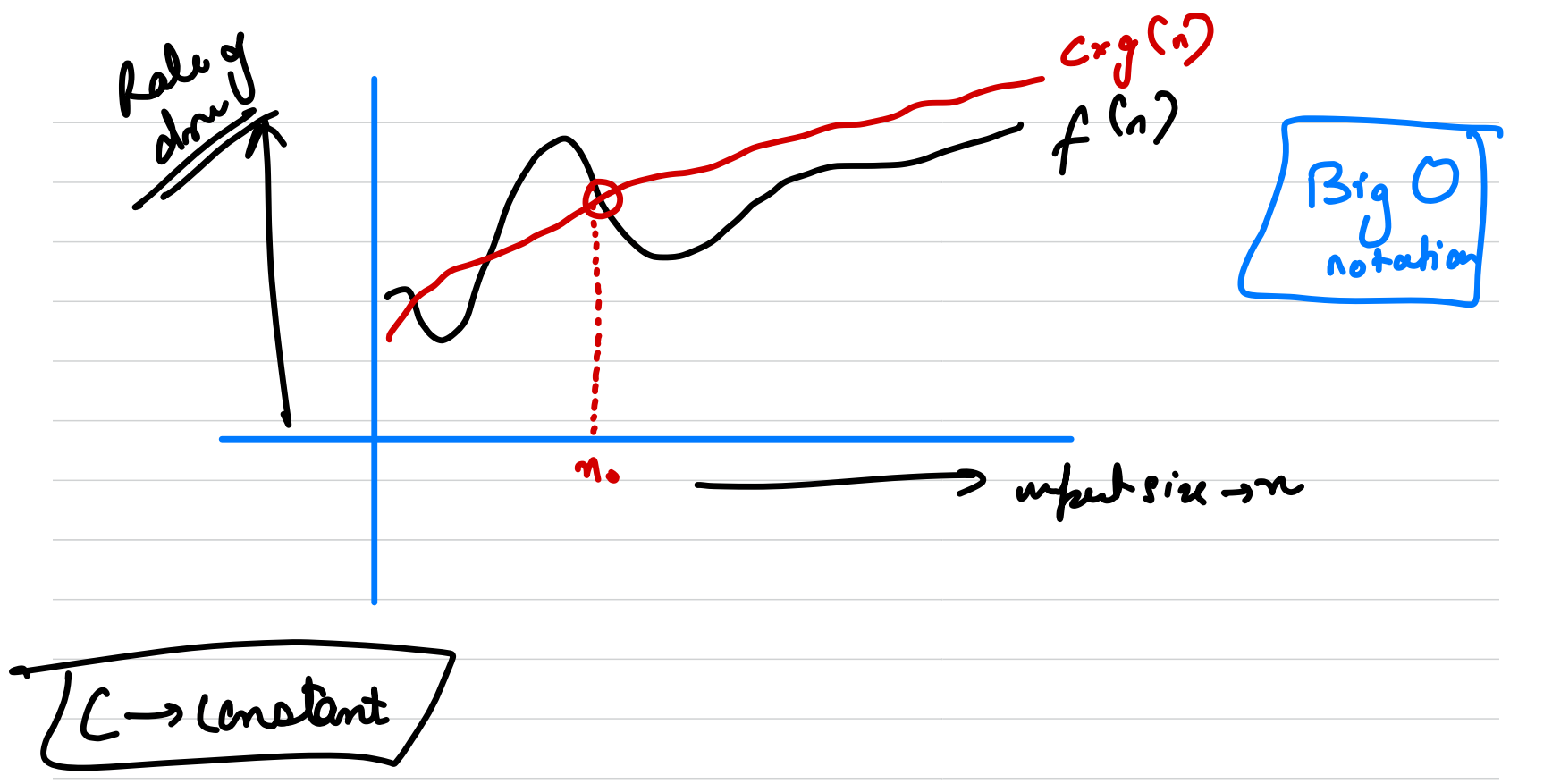

time taken
a_1
a_2
input size

Rate of growth ?? → Rate at which the running time increased as a func of input

$y = f(x)$

function take input size as argument


Best Case
Avg Case
Worst Case
asymptotic analysis


Rate of growth
c*g(n)
f(n)
Big O notation
n0
input size → n
C → Constant

Big O notation ⇒ gives the tightest upper bound of a function

→ $f(n) = O(g(n))$ ←

$f(n) = n^4 + 99n^2 + 35n + 6$

$g(n)$ ⇒ gives the max rate of growth for $f(n)$ at large values of 1

$g(n) = n^4$

$O(g(n)) = \{ f(n)$: there exist +ve constant c and n_0 such that

$0 \leq f(n) \leq cg(n) \quad \forall \; n \geq n_0 \}$

↳ tightest upper bound

$f(n) = n^2 + 20$

$g(n) = n^2$

$c = 2$

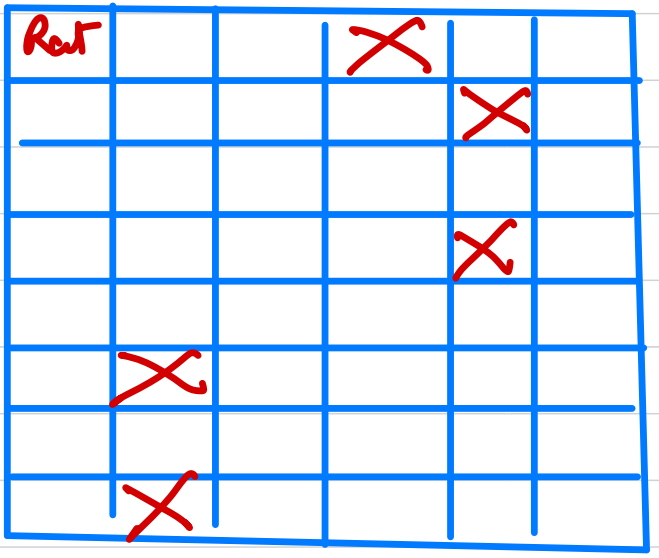
$f(100) = (100)^2 + 20 \rightarrow 10020$

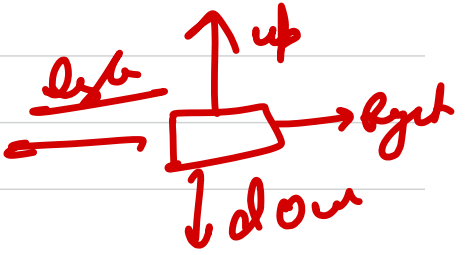
$cg(100) = 2 * (100)^2$

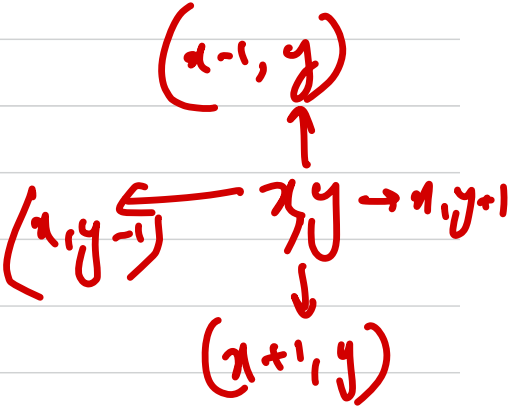
↳ 20000

Example → Rat in a maze

Total ways such that Rat reaches from TL to BR







Time Complexity → Tightest upper Bound

$f(x,y) = f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1)$

return the no. of ways in which you can reach from (x,y) to BR

$4n^2$ ← good upper bound

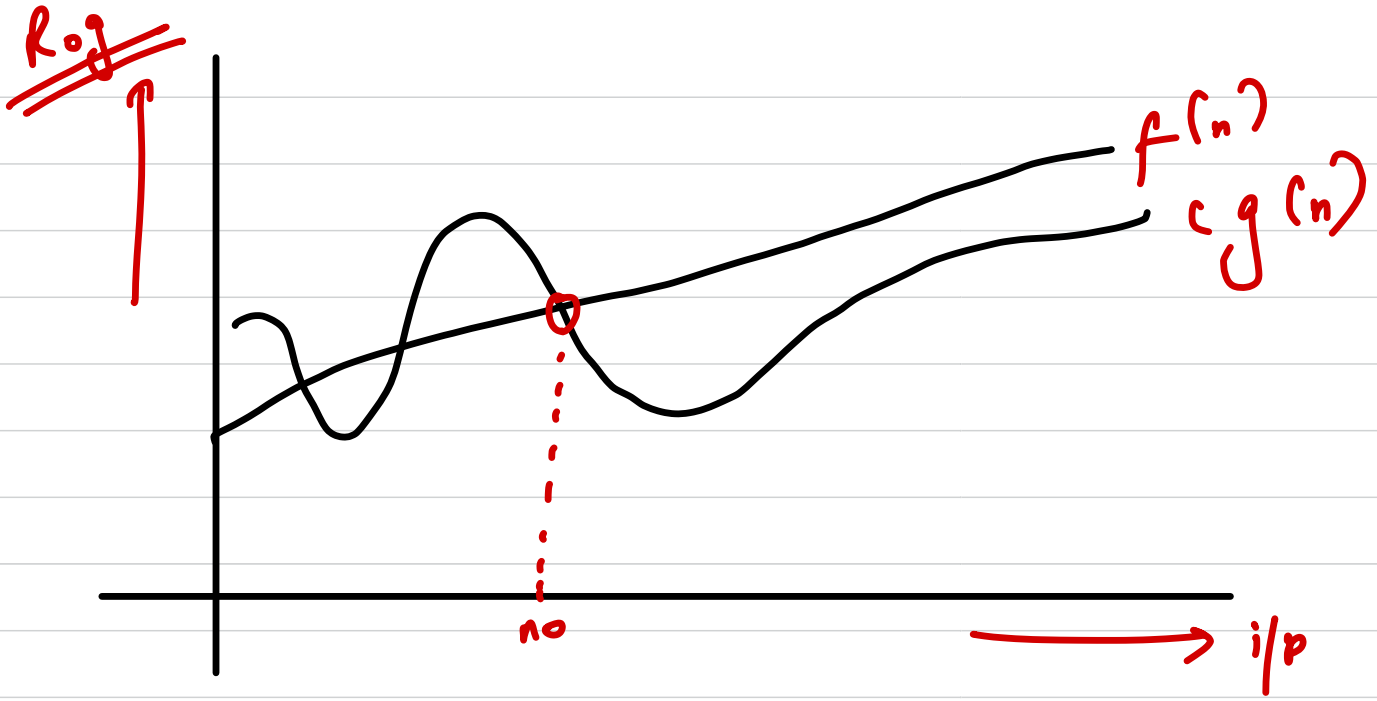
$3n^2$

$2 \left(\text{no. of boundary cells} \right) \times$

nothing is possible < tight bound < everything is possible

So a loose bound

little o notation



Tightest lower bound

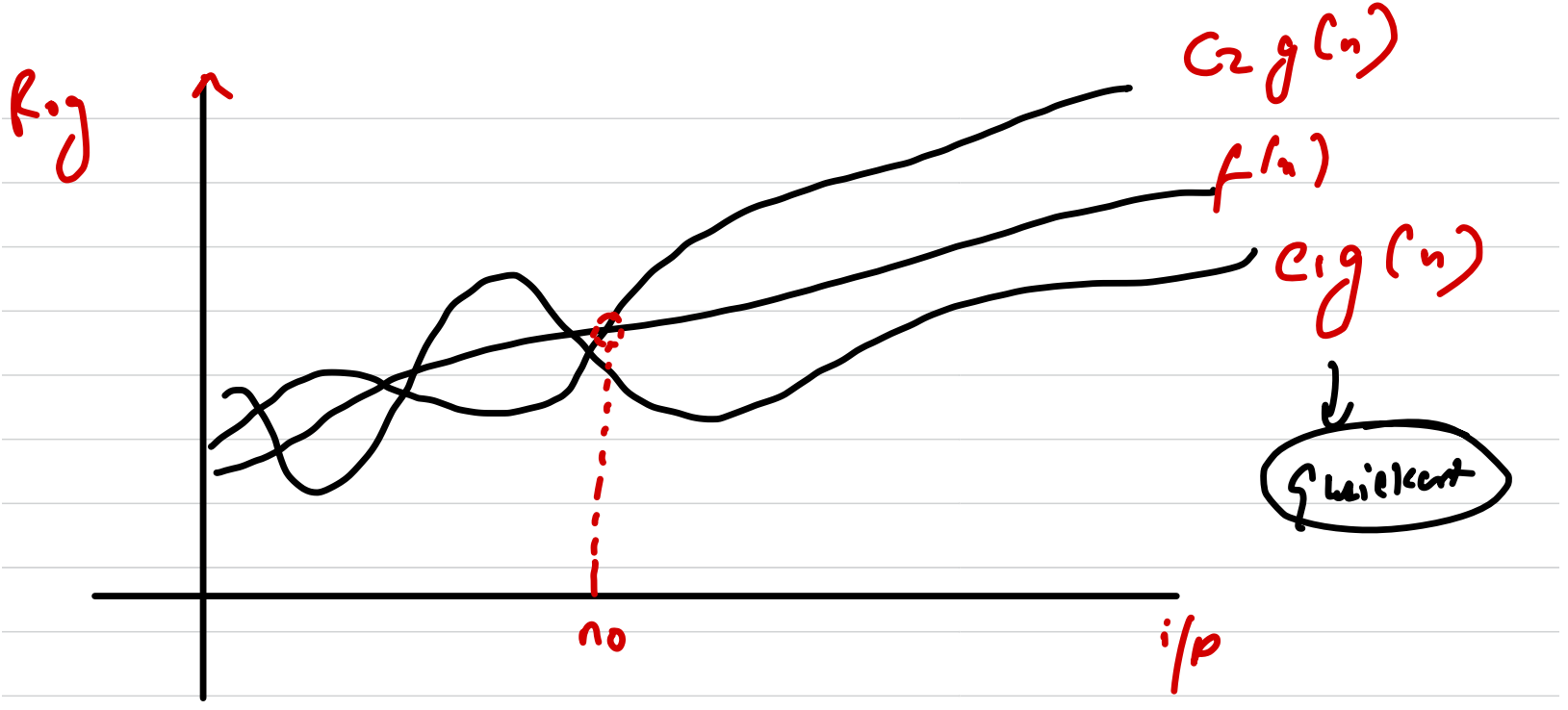
$f(n) = \Omega(g(n))$

$0 \leq cg(n) \leq f(n)$

$$f(n) = 100n^2 + 10n + 60$$

$$g(n) = n^2$$

$$C = 100$$



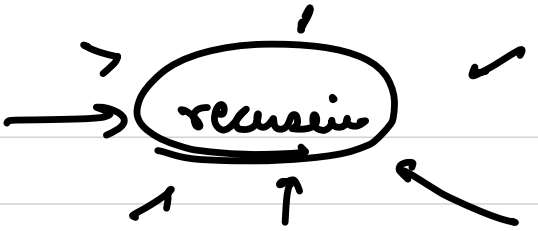
Average case (Θ) (theta)

$f(n) = \Theta(g(n))$

$c_1 g(n) \le f(n) \le c_2 g(n)$

$\forall\ n \ge n_0$

→ iteration



$f_n = f_{n-1} + f_{n-2}$

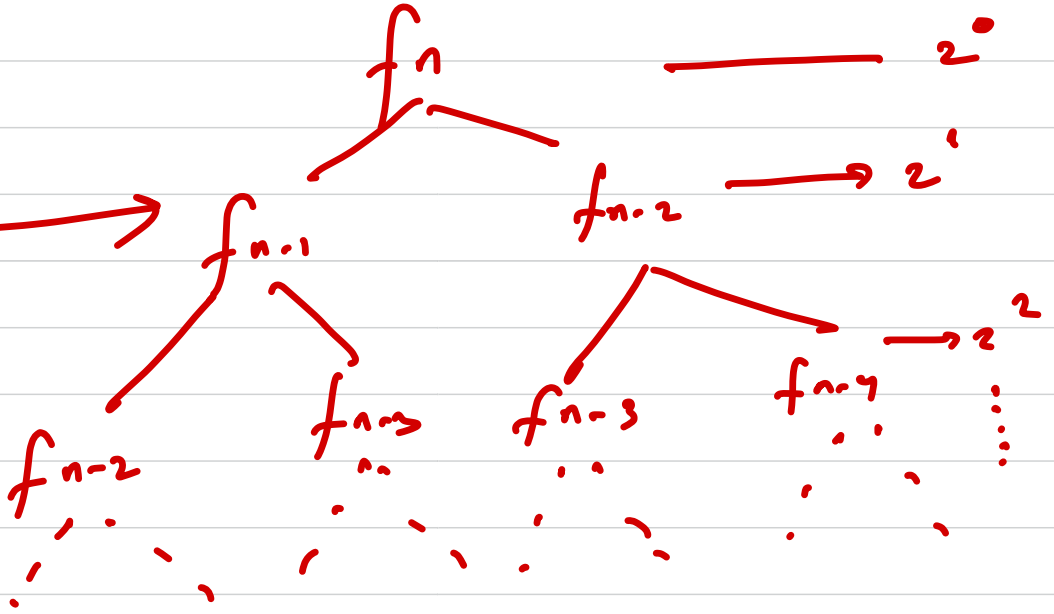
$f_n = n \times f_{n-1}$

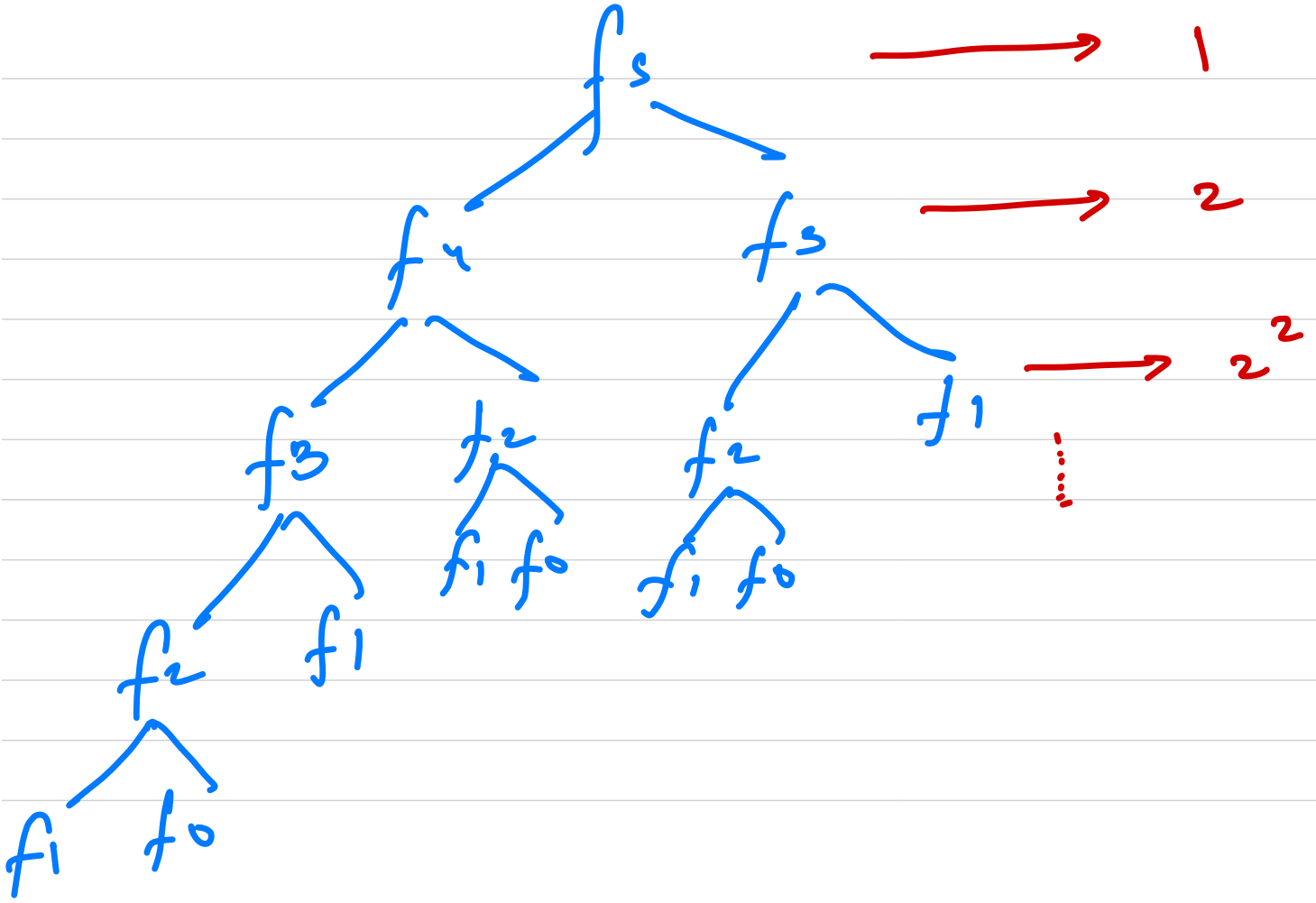
Recursive

fibonacci → $f(n) = f(n-1) + f(n-2)$

Higher upper bound for the func

Recurr Tree




f5
f4
f3
f3
f2
f2
f1
f2
f1
f1
f0
f1
f0
f2
f1
f1
f0
1
2
2^2

$2^0 + 2^1 + 2^2 \ldots\ldots\ldots 2^{n-2}$ → $x+y$ → $1 \times (2^{n-1}-1)$

GP → r → ratio

$r > 1$ $\quad$ $r < 1$

a – Constant

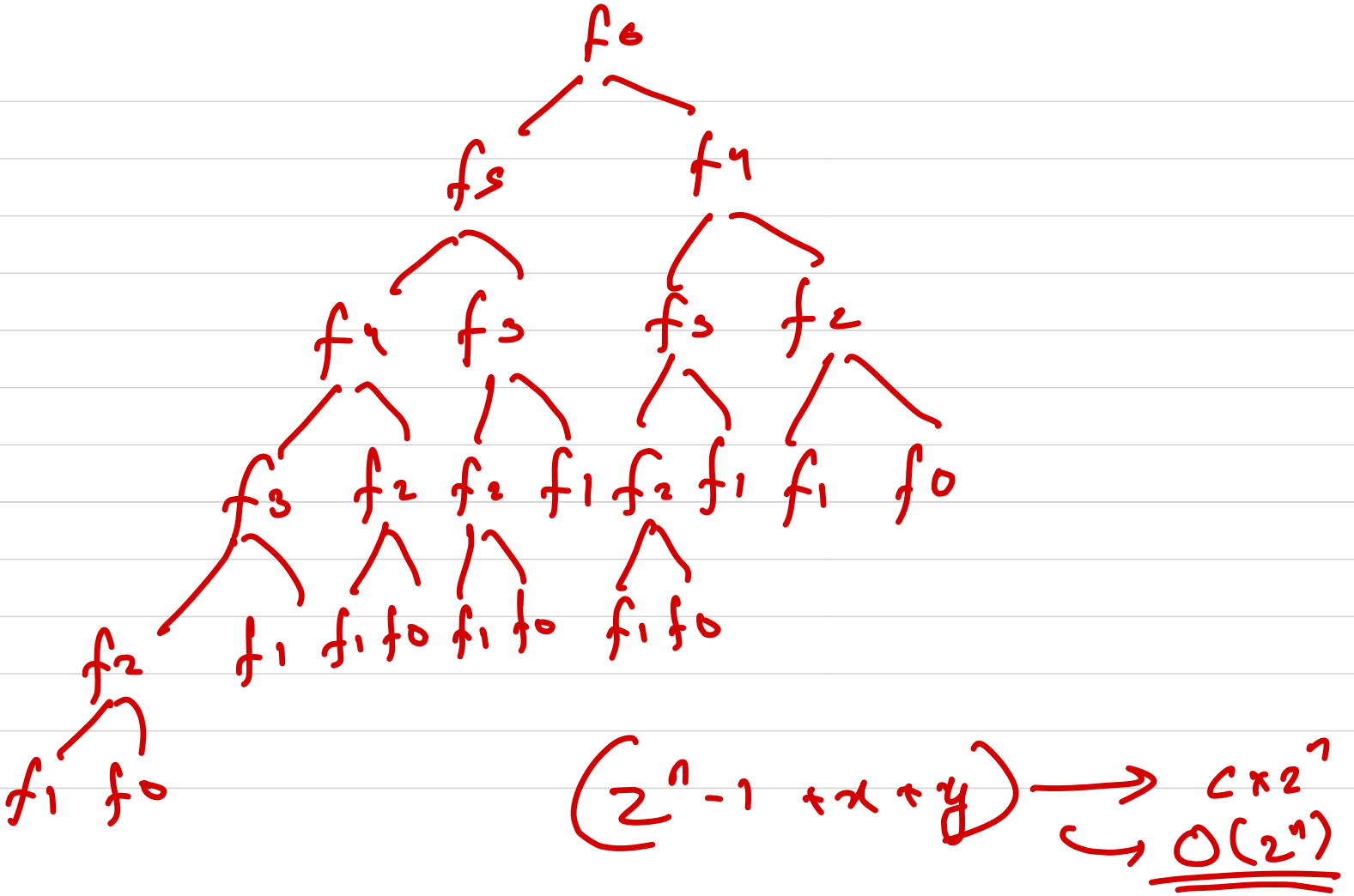
$a \quad ar \quad ar^2 \ldots\ldots\ldots ar^{n-1}$

↳ last term

Sum → $a(1 + r + r^2 - - - - - r^{n-1})$ → last term

$a \times \frac{1 \times (r^n - 1)}{r-1} \Rightarrow \frac{a(r^n-1)}{r-1}$ $\quad r > 1$

$\frac{a(1-r^n)}{1-r}$ $\quad r < 1$


f6
f5
f4
f4
f3
f3
f2
f3
f2
f2
f1
f2
f1
f1
f0
f2
f1
f1
f0
f1
f0
f1
f0
f2
f1
f0
(2^n - 1 + x + y)
c x 2^n
O(2^n)

C → const

$$\Rightarrow f(n) = 2f\left(\frac{n}{2}\right) + cn$$

$$f(n) = 3f\left(\frac{n}{2}\right) + cn^2$$

$$\Downarrow$$

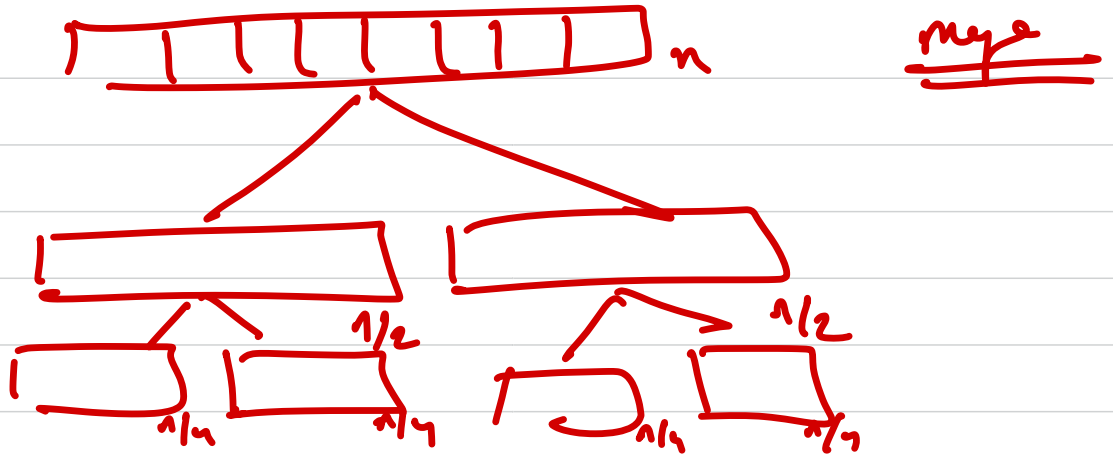
general eq^n of D n C

$$f(n) = a\,f\left(\frac{n}{b}\right) + h(n)$$

$a \rightarrow$ no. of subproblem

$\left(\frac{n}{b}\right) \rightarrow$ size of new subproblem

$h(n) \rightarrow$ polynomial fun^n of n



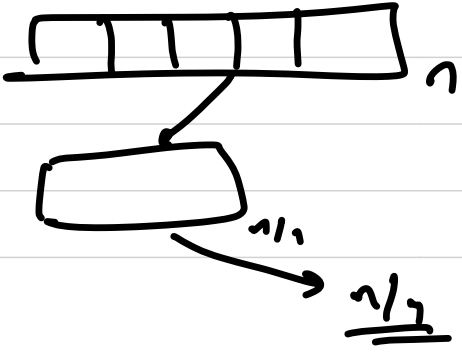
$$MS(n) = MS(n/2) + MS\left(\frac{n}{2}\right) + merge\left(\frac{n}{2}, \frac{n}{2}\right)$$

$$f(n) = f\left(\frac{n}{2}\right) + c$$

$a = 1$

$\frac{n}{b} \to \frac{n}{2}$

$h(n) = cn^0$



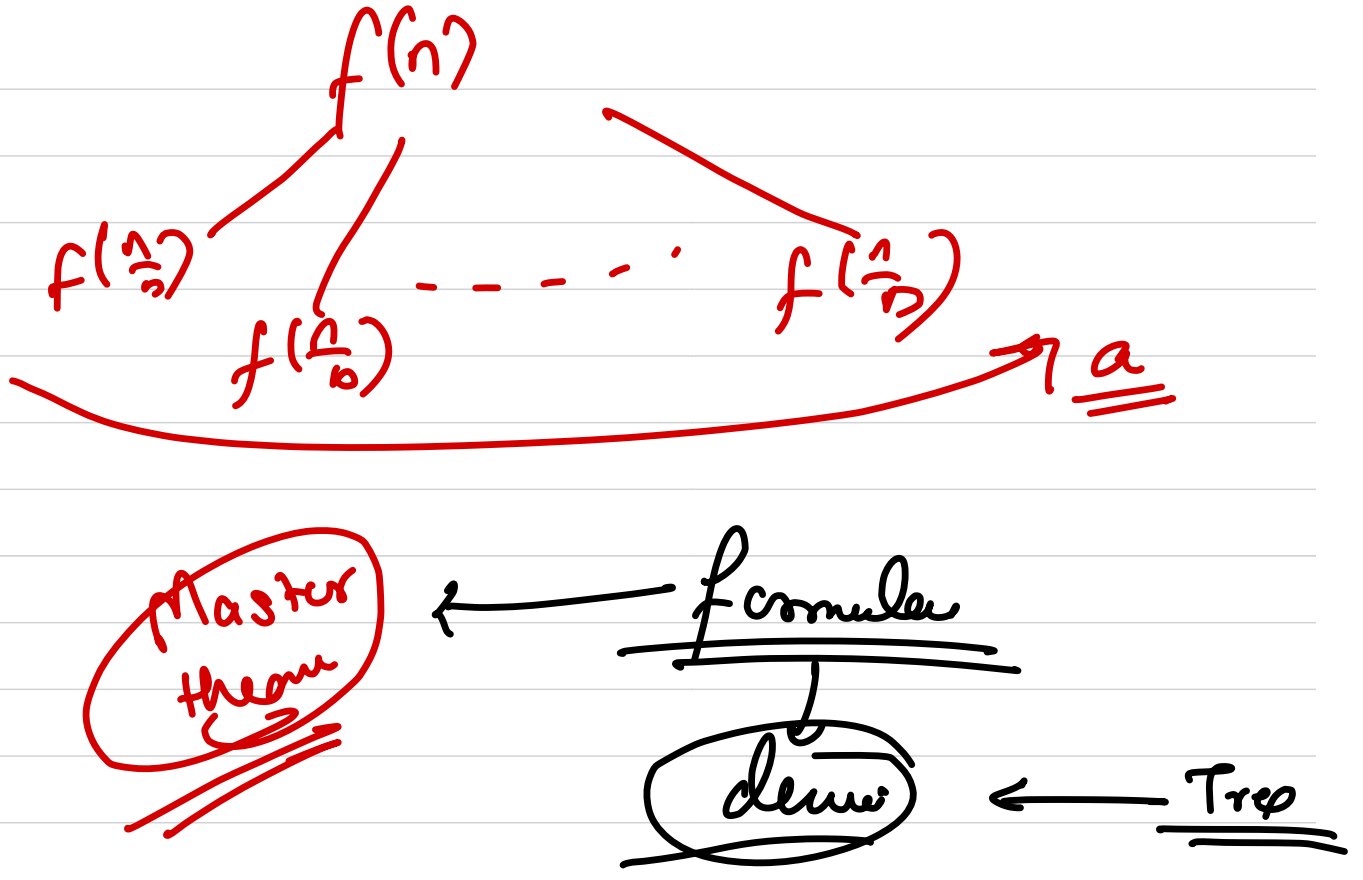
$$f(n) = a\,f\left(\frac{n}{b}\right) + h(n)$$

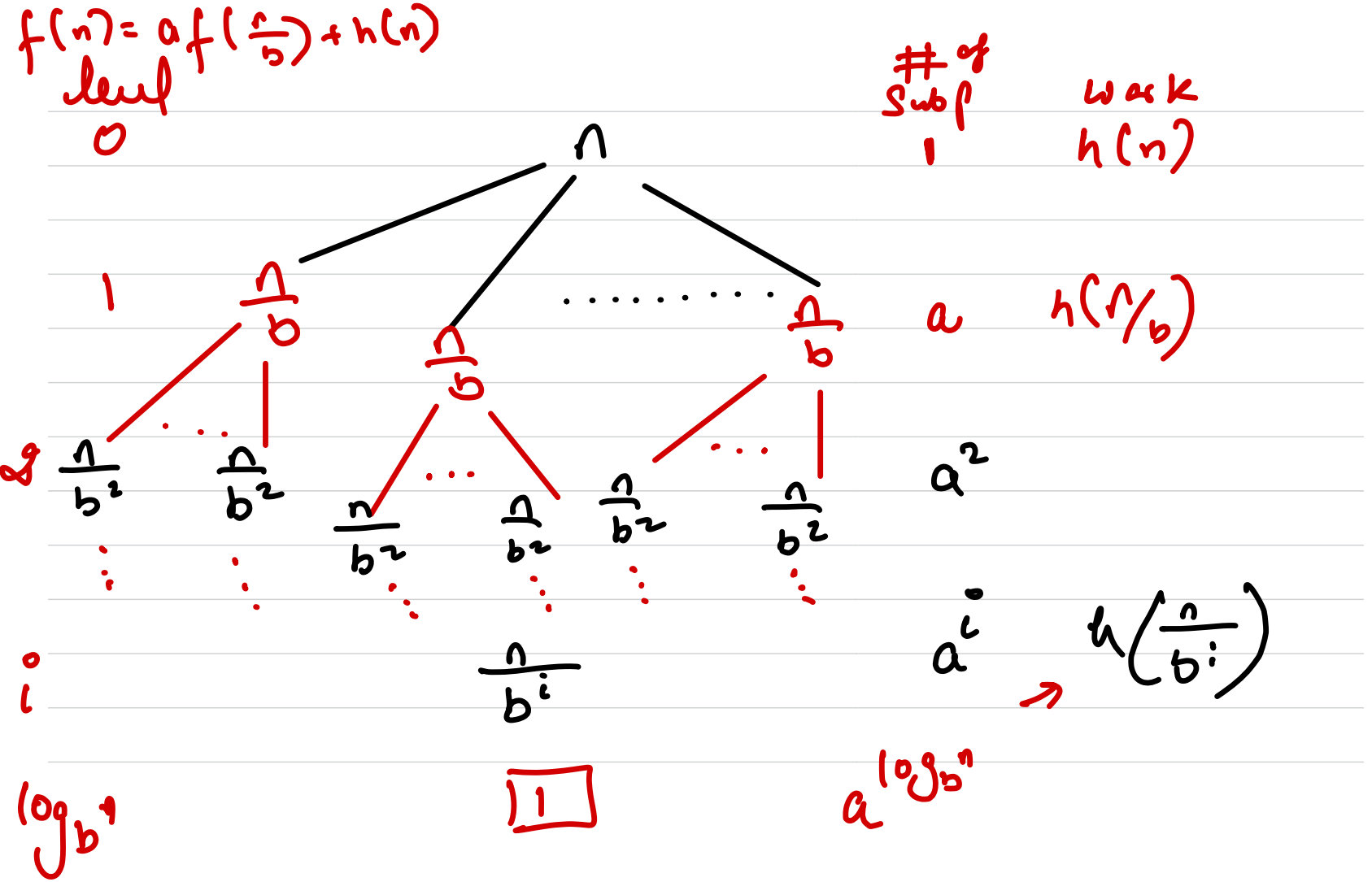
→ general eq^n of DnC

multiple approach

- → Recursn Tree
- → Master Theorem
- → akra bazi formula
- → assumption
- → substitn

} ←


f(n)
f(n/a)
f(n/b)
f(n/b)
a
Master theorem
formula
demo
Trop


$f(n) = af(\frac{n}{b}) + h(n)$
level
# of subp
work
0
1
$h(n)$
1
$\frac{n}{b}$
a
$h(\frac{n}{b})$
2
$\frac{n}{b^2}$
a^2
i
$\frac{n}{b^i}$
a^i
$h(\frac{n}{b^i})$
$\log_b n$
1
$a^{\log_b n}$

a^i

$n \rightarrow \frac{n}{b} \rightarrow \frac{n}{b^2} - - - - - \cdots \frac{n}{b^k}$

$\frac{n}{b^k} = 1$

$n = b^k$

$k = \log_b n$

$$a^i \times h\left(\frac{n}{b^i}\right)$$

$h(n) \rightarrow$ polynomial in n

$h(n) = n^k$

$h\left(\frac{n}{b^i}\right) = O\left(\frac{n}{b^i}\right)^k$

$a^i \times O\left(\frac{n}{b^i}\right)^k \Rightarrow O(n^k)\left(\frac{a}{b^k}\right)^i$

Total $\Rightarrow \sum_{i=0}^{\log_b n} O(n^2)\left(\frac{a}{b^2}\right)^i \rightarrow$ GP

if $\frac{a}{b^2} < 1$ i.e $r < 1$

$a, ar, ar^2 \text{ ------ } ar^{n-1}$

$r < 1 \rightarrow ar\left(\frac{1-r^n}{1-r}\right) \rightarrow O(a)$

$r > 1 \rightarrow \frac{a(r^n-1)}{r-1} \rightarrow O(ar^n)$

$r = 1 \rightarrow O(na)$

Bloomberg

$\frac{a}{b^k} < 1 \longrightarrow O(n^k)$

$\frac{a}{b^k} > 1 \longrightarrow O\left(O(n^k)\left(\frac{a}{b^k}\right)^{\log_b n}\right)$

$x^{\log_y n} \to n^{\log_y x}$

$x^{\log_y x^n}$

$\Rightarrow n$

$\longrightarrow O\left(O(n^k)\ \frac{a^{\log_b n}}{b^{k \log_b n}}\right)$

$\longrightarrow O\left(O(n^k)\ \frac{n^{\log_b a}}{n^k}\right)$

$\longrightarrow O(n^{\log_b a})$

$\frac{a}{b^k} = 1 \rightarrow O((\log_b n + 1) \times O(n^k))$

$\rightarrow O(n^k \log_b n)$

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

$a = 3$ $\quad b = 4$

$n^k =$ $\quad k = 1$

$\frac{a}{b^k} \rightarrow \frac{3}{4} < 1$

$O(n \log n)$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$a = 4$ $\quad$ $k = 2$

$b = 2$

$$\frac{a}{b^k} \Rightarrow \frac{4}{2^2} \Rightarrow \frac{4}{4} = 1$$

$$O(n^2 \log_2 n)$$

$$\hookrightarrow O(n^2 \log n)$$

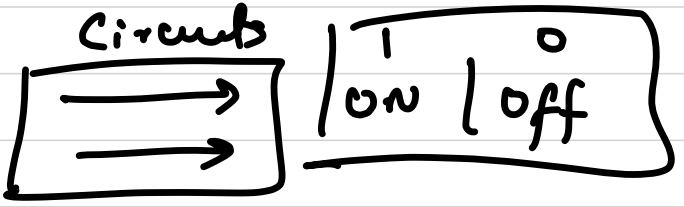
Bit manipulation

Huffman encoding

1 2 → computer → 2 2 (0 -1) ← understandable

Binary System

Circuits



Encryption & data compression purpose

Bits → 0 -1

int / float / char ⟷ bytes

Bit manipulation refers to the technique where we use bits inorder to solve complex problems

Bitmasking ⟷ Bits + mask → to hide/cover something

In bit masking we use bits to mask/cover/hide some data into bits. or we can say we represent some data using only bits.

Ex $S = \{1, 2, 3\}$ → 5 101

1 ✓ → taken

0 → not taken

$x \in$ Subsets (S)

$x \rightarrow$ [101] → int → O(1)

S → $\{1, 3\}$ → array → O(n)

$x \Rightarrow 001$ → $\{3\}$

$x \Rightarrow 000$ → $\{\emptyset\}$

1) 14_{10} ⟶ 1110_2

Base → Decimal

$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

The main tool to implement Bit manipulation is

Bitwise operators

In C++ we have one more tool → BitSet

⇒ And (&)

Bitwise

2 & 3 → 2

(10) & (11)

10
11

10 → 2

x and y

and

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

2 && 3

Tom && Tom

different than && and logic gate

logical

Boolean logic

logic

Boolean

0/1

0/1

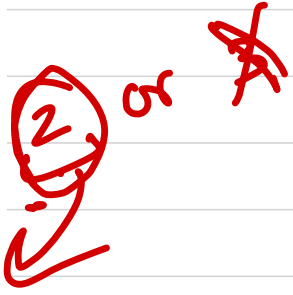
0/1

B-C && B-C

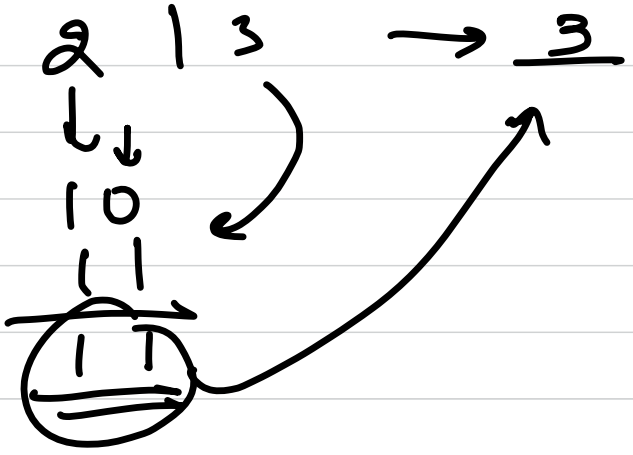
Tom → 1

x	and	y	z
0		0	0
0		1	0
1		0	0
1		1	1

x	or	y	z
0		0	0
0		1	1
1		0	1
1		1	1



Or (|) → Bitwise



Xor (^)

5 ^ 3 = 6

(101) ^ (011)

```
  1 0 1
  0 1 1
-------
  1 1 0   → 6
```

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

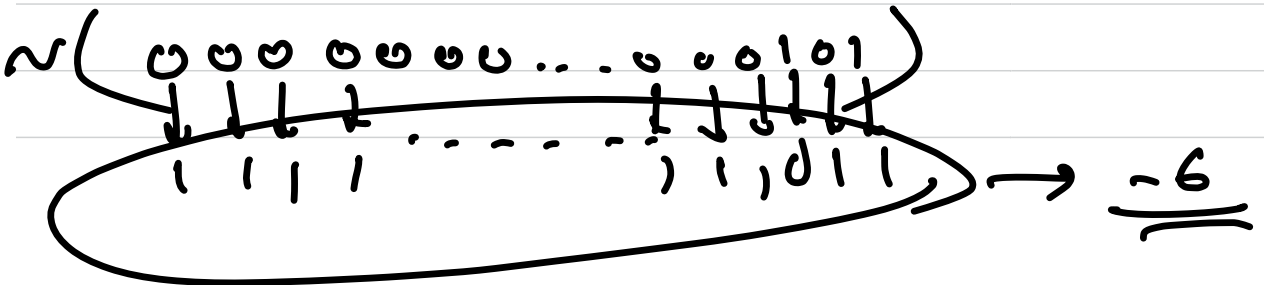
NOT (~)

x	~x
1	0
0	1

~5 → ~(101) → (010)$_2$ = 2

How -ve is stored in computers

↳ 2's compliment



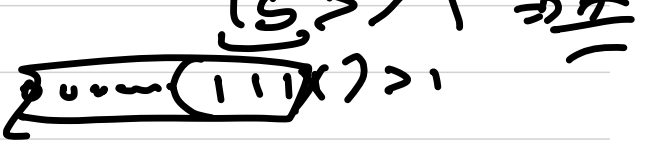
Left Shift (<<)

x<<y → appended y zeroes at the end of x and trims the first y bits.

5<<2 → steps

(101) << 2 → 10100

Right Shift (>>) x >> y



↳ prepends y zeroes at the start of x & trims the last y zeroes.

Q. Calculate n^{th} power of 2

O(1)

$2^0 - 1 \longrightarrow 00\ldots.001$

$2^1 - 2 \longrightarrow 000\ldots.010$

$2^2 - 4 \longrightarrow 000\ldots.100$

$2^3 - 8 \longrightarrow 000\ldots.1000$

$2^4 - 16 \longrightarrow 000\ldots.10000$

$2^5 - 32 \longrightarrow 00\ldots.100000$

2^x

$\Rightarrow$ 1<<x

If we want to divide a no.

$(x >> 1)$

$S \rightarrow 101^{x}0\,10$

$S/2 \rightarrow 2$

$\frac{n}{2^m} \Rightarrow n >> m$

$\frac{20}{2^3} \rightarrow \ldots(10100) >> 3$

00010

$\left\lfloor \frac{20}{8} \right\rfloor = 2$

Q. Check if a number n, is a power of 2?

⇒ We know every 2^x has only one set bit.

So count the no. of set bits. And if it is 1 then number is power of 2.

⇓

How to count no. of set bits

(00...0101101011)$_2$

32

```
for(i=0;i<32;i++)
  if((x>>i)&1)==1)
    set ++ count
  else
    not set
```

Brute force

Iterate over all the bits and check if the i^{th} bit is set or not?

O(no. of bits)

(How to check i^{th} bit is set or not??)

x (5 4 3 2 1 0) $10110\underline{0}_2$

20 → 10100
19 → 10011

→ bit is set or not

$x \,\&\, 1$

$10110\underline{0}$
$\&\ 000001$
$\rightarrow 000000 \rightarrow 0$

$(x >> 1) \,\&\, 1 == 0 \rightarrow$ not set

$01011\underline{0}$
000001
000000

for any i^{th} bit $\rightarrow$ $(x >> i) \,\&\, 1$

Efficient approach to count the no. of set bits.

Brian Kernighan Algo

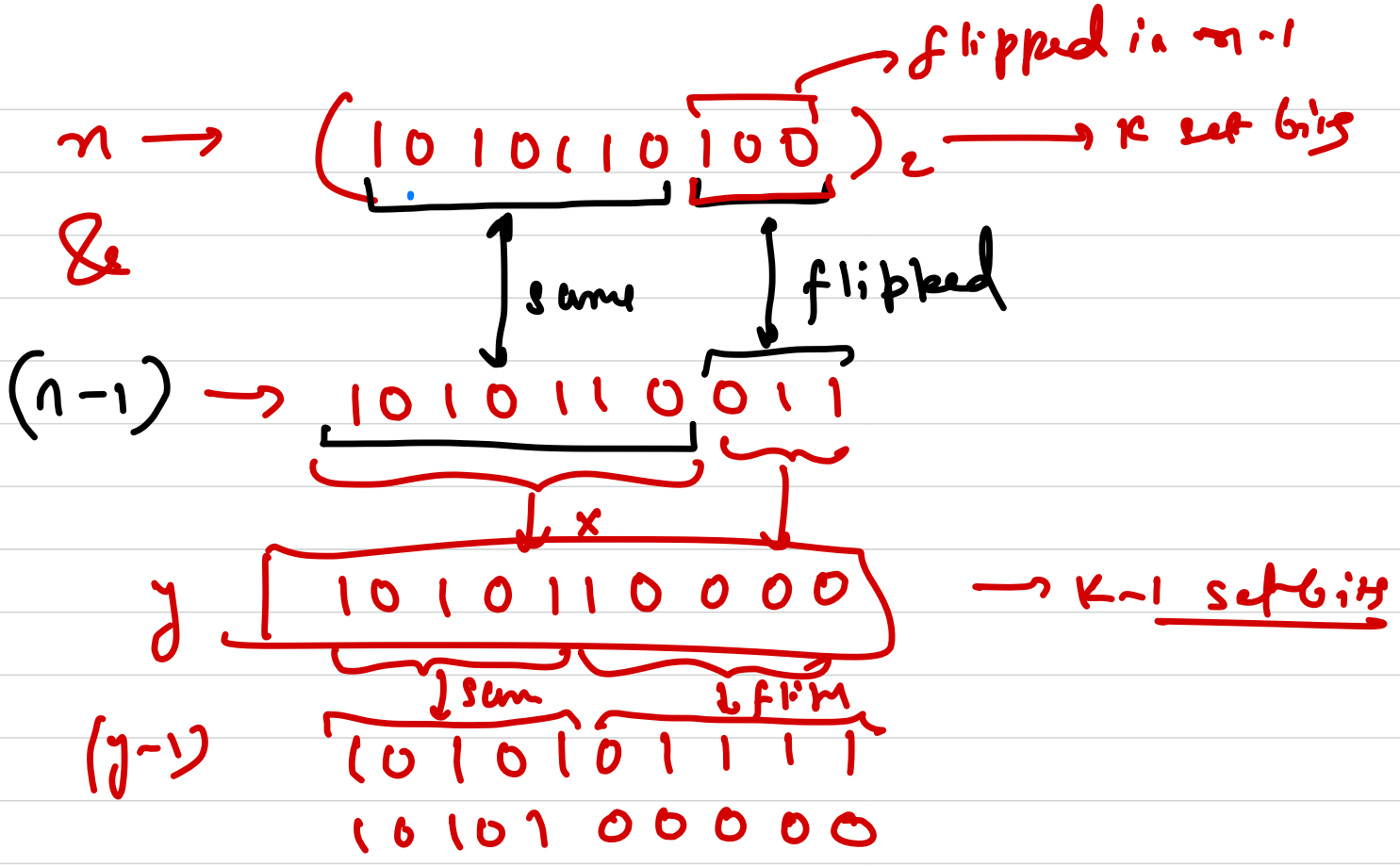
flipped in n-1

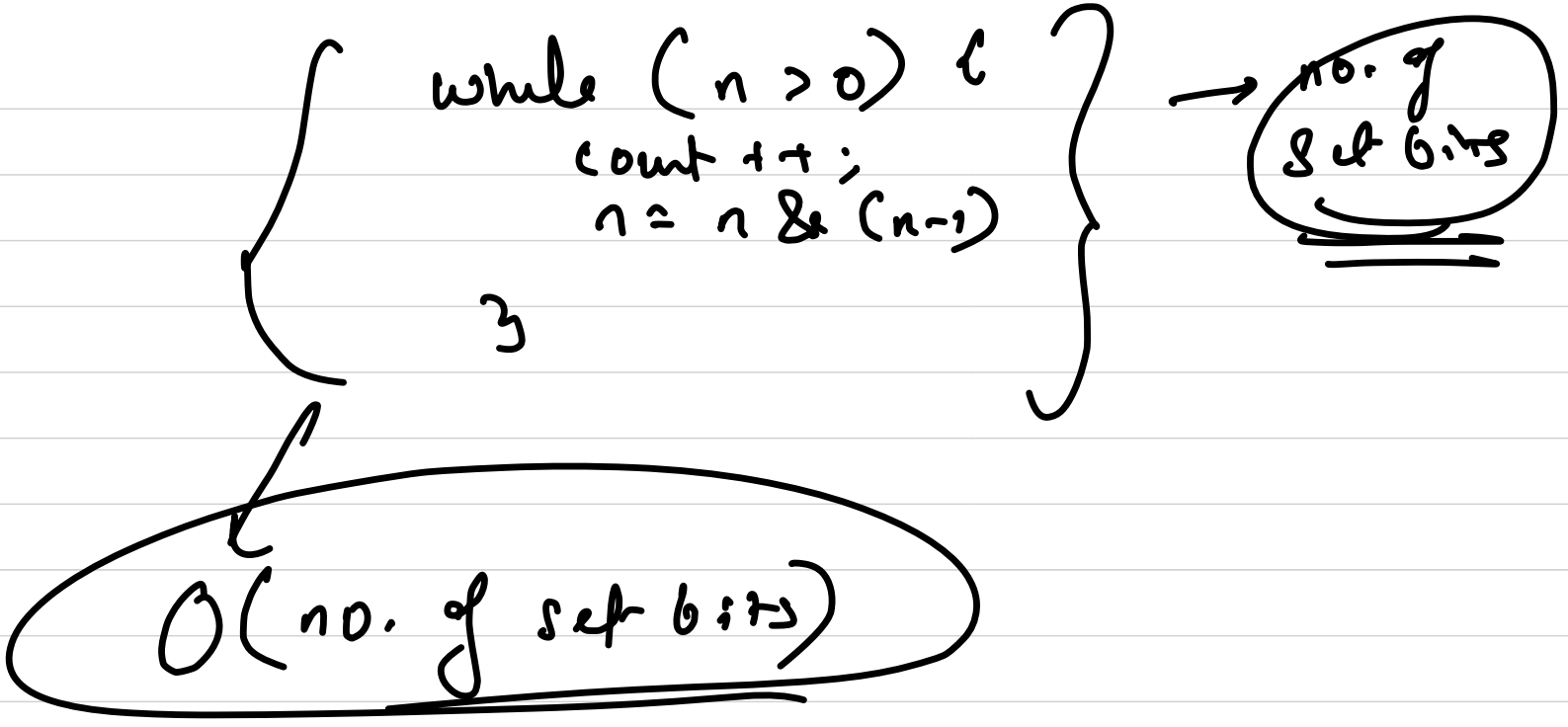
$n \rightarrow (10101\overline{01100})_2$

There is an observation

$(n-1) \rightarrow 1010110011$

in (n-1) → we have every bit flipped starting from last set bit of n.


flipped in n-1
n →
(10101010100)₂
→ k set bits
&
same
flipped
(n-1) →
10101100011
x
y
10101100000
→ k-1 set bits
same
flipped
(y-1)
10101011111
10101000000


while (n > 0) {
count++;
n = n & (n-1)
}
no. of set bits
O(no. of set bits)

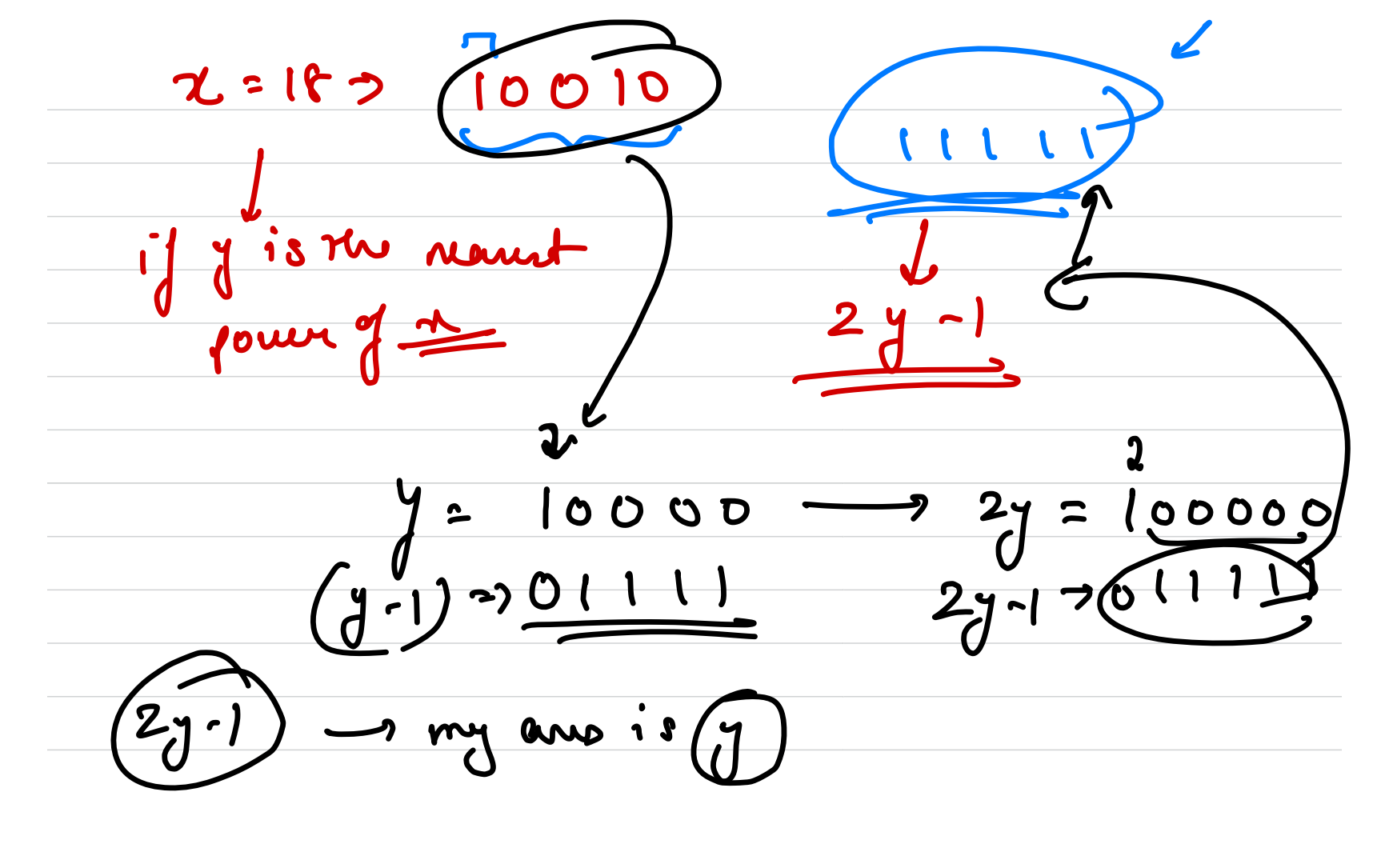
Q. Find biggest power of 2 which is less than or equal to N.

N = 18 → 16

18 → 10010

$x \rightarrow (1111111)_2 \rightarrow (2^i - 1)$ ← Form

↓ if all bits are 1


x = 18 → 10010
if y is the nearest power of x
2^y
y = 10000
(y-1) → 01111
11111
2y - 1
2y = 100000
2y-1 → 01111
2y-1 → my ans is y

x → how to count all right bits of MSB to 1.

N ⇒ 10000

N = N | (N >> 1) ⇒

```
 10000
  1000
 11000
```

= N

N = N | (N >> 2) ⇒

```
 11000
   110
 11110
```

⇒ N

N = N | (N >> 4) ⇒

```
 11110
     1
 11111
```

= N

4 byte (32)

$N = N | (N >> 1)$

32 Bits

$N = 1001\ 1001\ 0111\ 0011$

$N = (N | (N >> 1)) \Rightarrow 11\ \_\_\_\_\_$

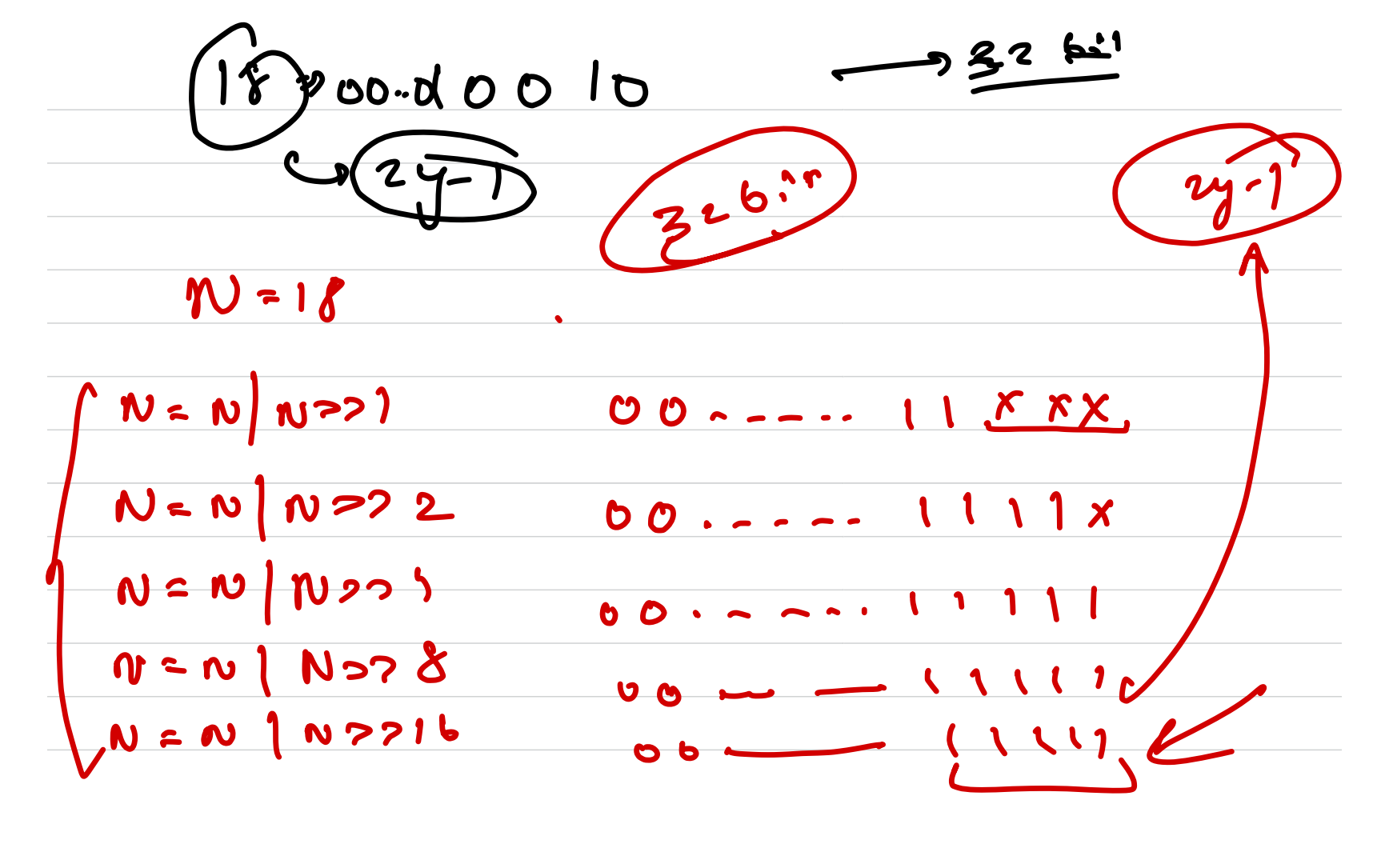
$N = (N | (N >> 2)) \Rightarrow 1111\ \_\_\_\_\_$

$N = (N | (N >> 4)) \Rightarrow 1111\ 1111\ \_\_\_\_$

$N = (N | N >> 8) \rightsquigarrow 1111\ 1111\ 1111\ 1111$

$N = (N | (N >> 16) \rightsquigarrow (1 \ldots\ldots 1)_2$

32 bits


18
00..000 10
2y-1
32 bit
26:31
2y-1
N = 18
N = N | N>>1
N = N | N>>2
N = N | N>>4
N = N | N>>8
N = N | N>>16
00..... 11 xxx
00..... 1111x
00..... 11111
00..... 11111
00..... 1111

64 bit

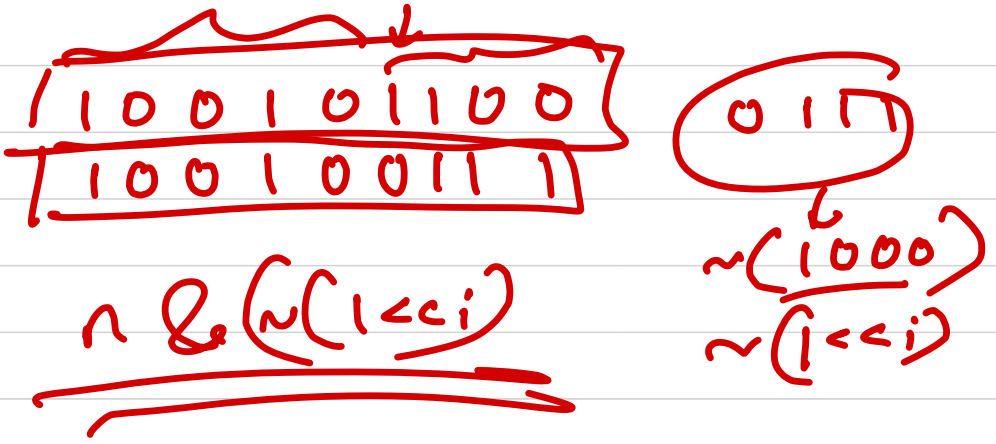
$2y - 1 \rightarrow 2$

$y = \frac{(2+1)}{2}$

$(a_n + 1) >> 1$;

```
int poww2-round(int x) {
    x = x | (x>>1);
    x = x | (x>>2);
    x = x | (x>>4);
    x = x | (x>>8);
    x = x | (x>>16);
    return (x+1)>>1;
}
```

Q: How to unset any i^{th} bit.



Q. Given a number x, return a no. which has only rightmost set bit of x.

x : 1 0 0 | 1 0 0

ans → 0 0 0 1 0 0

(x-1) → 1 0 0 0 1 1

x & (x-1) ⇒ 1 0 0 | 0 0 0

x ^ (x & (x-1))

ans