Q ⇒ why to analyse an algorithm??

→ whatever algo you wrote it should be efficient in terms of execution time & space taken.

SSD/HDD → Secondary Storage

RAM, cache → primary Storage

CPU / GPU → process unit

↓

clock speed

boosting

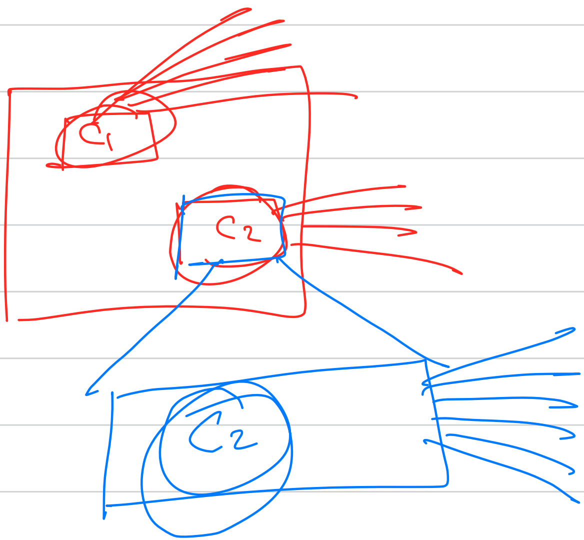
thread

core

Experimental analysis

[when you start code execution, note the time & when you end it note it again, & see the difference

1sec → 10^8 cycles by cpu

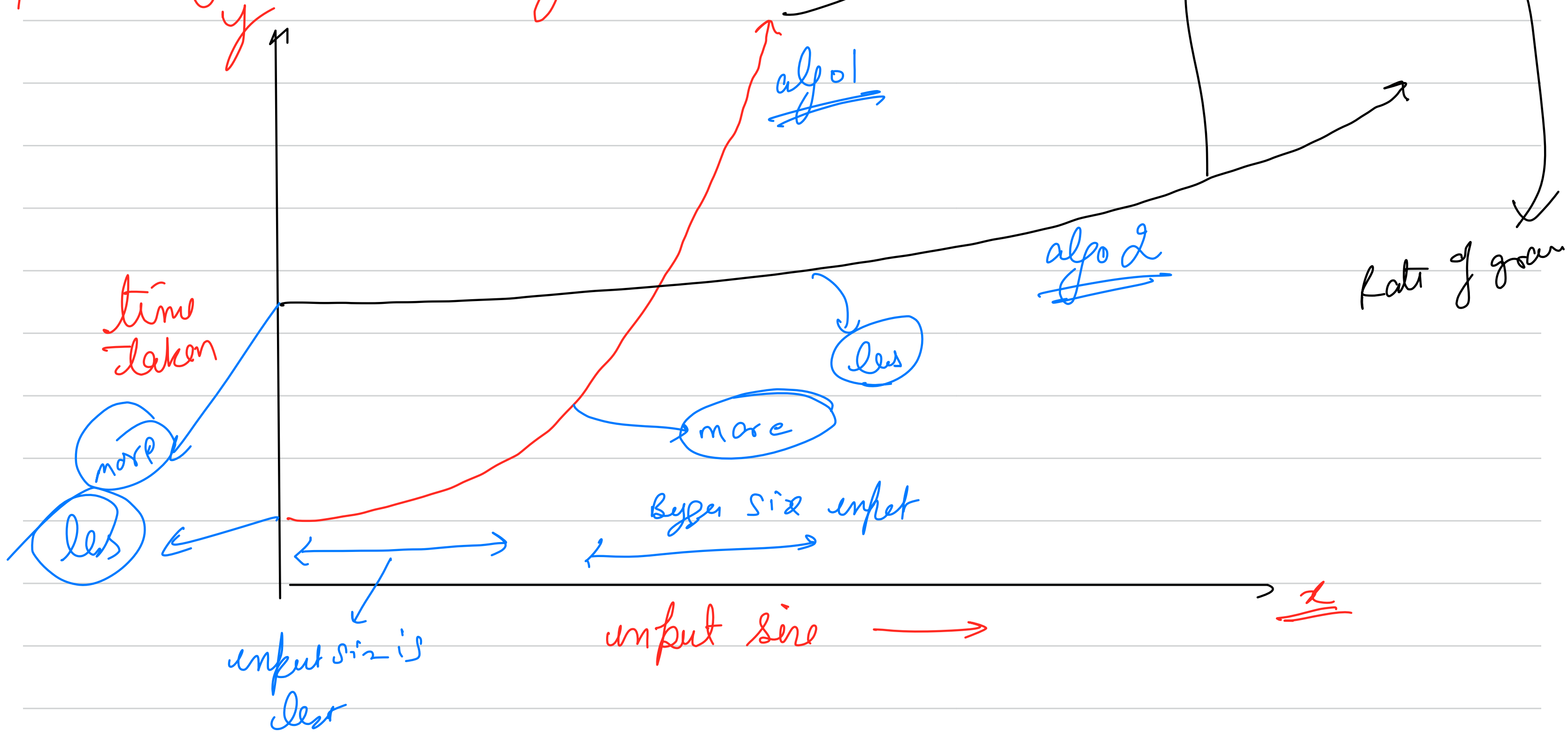


↳ waiting state
↓
core picks any one → executes some part of it → transfer it to waiting state again → picks another one

curve is always an ~~asy~~ asymptote

we deal with time taken by algo, w.r.t input constraint → polynomial

Asymptotic analysis



no. of operat → no. of lines execute,

Same algo when given inputs of varying size will execute different no. of operation

10 1s

Analysis based on input given

We will not focus on individual time value instead we will focus on Rate of growth

Rate of growth is the rate at which runtime of an algorithm increases

highest degree here decides the rate

$f(n) = n^3 + 2n + 1$

$g(n) = n^2 + 99n + 100$

which one will grow faster or have a higher rate

$n = 100$

$n^3 \Rightarrow 10^6$

↳ the rate of growth of a function mostly depends on the highest degree term.

$f(x) = 3$

→ exponent → a^n → a lot of time for small input

→ n^3

→ n^2

→ $n\sqrt{n}$

→ $n \log n$

→ n

→ $\sqrt{n}$

→ $\log n$

→ const → it is most often

time increases

$f(n) = n^2 + 99n + 100$

$n = 0$

→ min value of this function ??

$f(n) = = 100$

for some input $n = 0$, $n = 1$, we have very less value

$n < 0$ $n > 0$ → we have very high value

Types of asymptotic analysis

funcⁿ

Best Case runtime → lower bound

Avg Case runtime → avg bound

Worst Case runtime → concerns why??

upper bound

lower <= avg <= upper

and we denote runtime as polynomial of n

$\rightarrow f(n) = n^2 + 5000$

$n == 0$

$f(n) = 5000 \rightarrow$ least value

Best case

$f(n) \approx n^2$

$n > 0$
$n < 0$

avg

worst

notations

$\underbrace{\text{Worst}}$ → Big $\underline{\underline{O}}$ (Oh)

Avg → Big Θ (theta)

Best → Big Ω (omega)

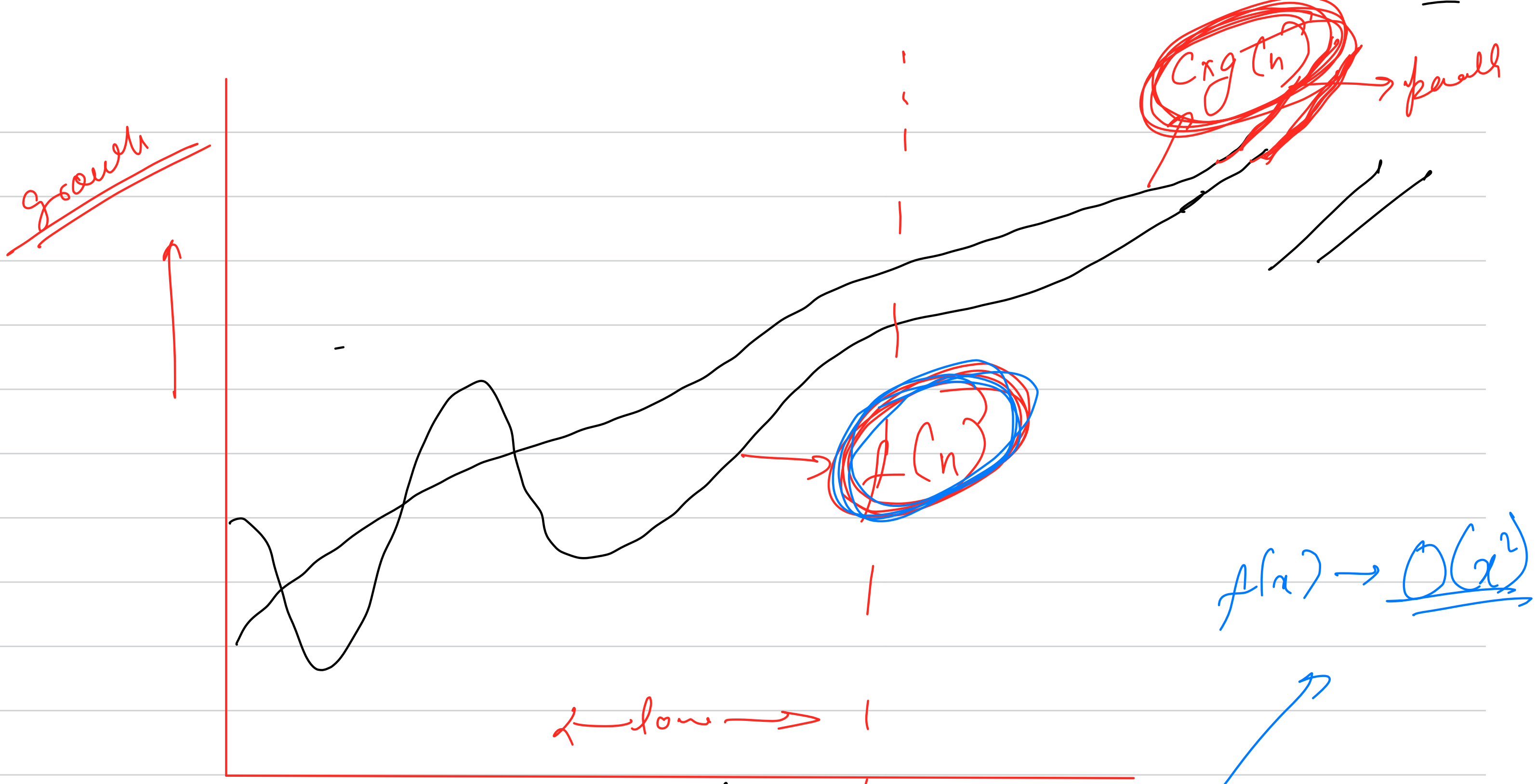
Big O $\rightarrow$ This notation gives us the tighest upper bound of a **function** i.e for a f^n $f(n)$ & for large values of n, the upper bound is denoted as the highest degree term.

insignificant

Ex $\underbrace{f(n)}= n^4 + \underbrace{3n^3 + 2n^2 + 1}_{\text{discard lower degree terms}}$; $g(n) = n^4$ | $\boxed{n \approx 10^7}$

Let's say for $f(n) \rightarrow g(n)$ gives the upper bound

That means $g(n)$ gives the max rate of growth for $f(n)$ at large values of n.



$f(x) \rightarrow O(x^2)$

g(n) → is an upper bound for f(n)

as bad as what → input size

$f(x) = 3x^2 + 3x + 1$

$g(x) = Kx^2$

$x >>>$

Ex

$f(n) = 3n + 8$

$3n + \cancel{8} \le 4n$

$\rightarrow O(n)$

$4n \rightarrow k_4$

Q. $f(n)$ $2n^3 - \underbrace{2n^2}$

$2n^3 - 2n^2 \le \boxed{2}n^3 \quad \forall\, n \ge 1$

$\boxed{2} \rightarrow$ const

$\rightarrow O(n^3)$

$f(n) = 2n^3 + 2n^2$

$2n^3 + 2n^2 \le 3n^3$

$g(n) = \boxed{3}n^2$

$\boxed{3} \rightarrow$ const

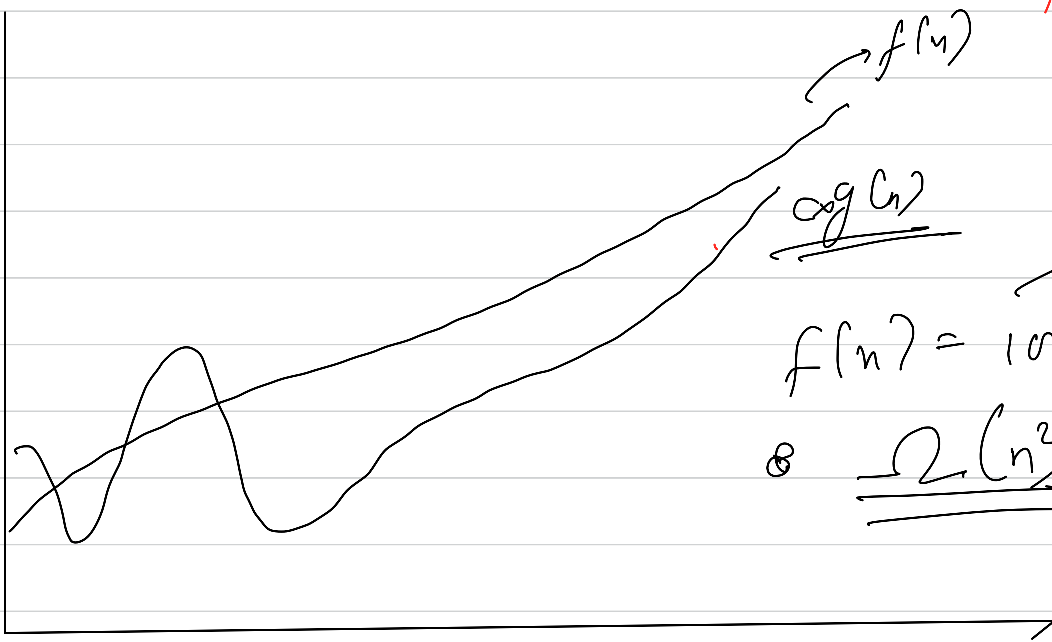
$\rightarrow O(n^3)$

Best Case (Ω) Big omega → tightest lower bound

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

large n

growth



→ input size (n)

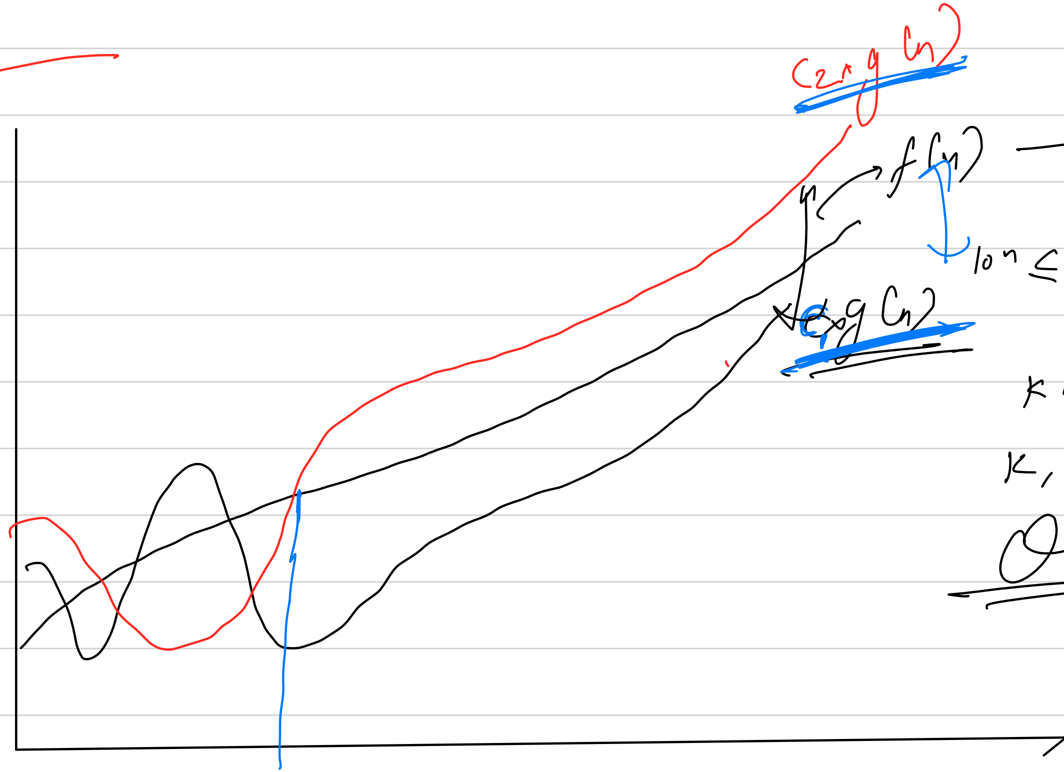
best case

$f(n) = 100n^2 + 10n + 50$

$\Omega(n^2)$

for large n

Avg case theta (Θ) notation → avg runtime



$f(n) = 10n + n$

$10n \le 10n + n \le 1\cancel{?}n$

$kn \le 10n + n \le mn$

$k, m \to$ const

$\Theta(n)$

Bubble sort algo

11, 1, -1, 2, 0, 1, 3, 4

11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Decreasing arr

10, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1

9, 11

→ n element

n^2 operations

1, 2, 3, 4, 5 → Best case (n) operation

Qs $\quad f(n) = \frac{n^2}{2} - \frac{n}{2} \longrightarrow$ Θ notation

$$\frac{n^2}{3} \leq f(n) \leq n^2$$

$$k \times n^2 \leq f(n) \leq k' n^2 \qquad \Theta(n^2)$$

multiple elements in a code → loops, nested loops, if else etc

① loops →

```
for (i=0; i<=n; i++) {
    ________
    ________
    ________
    ________
    ________
}
```

S statements

33 step

Total operation → $5n$

$33n$

$f(n) = 5n \longrightarrow O(n) \longleftarrow$ linear

$g(n) = n$

nested loops $\rightarrow$

for (i = 0; i <= m; i++) $\xrightarrow{m\ \text{times}}$

for (j = 0; j <= n; j++) { $\rightarrow$ n times

2 statements

}

$f(n) = 2 \times m \times n$

$O(mn) \rightarrow$ quadratic

$m == n$

$O(n^2)$

Outer loop

for (i = 0 ; i <= n ; i++) {

3 statements

}

$3n$

for (i = 0 ; i <= m ; i++)

4 statements

}

$4m$

$m == n$

Total operation $f(n) = 3n + 4m$ linear

$O(n + m)$ $\rightarrow O(n+n)$

$O(n)$

Q. if ( ) {

7 state — 7

} else {

3 state → 3

}

max (7, 3) → 7

$f(n) = \cancel{7}$ / 8 / 3

O(1) → constant

a) $O(n)$

b) $O(n^2)$

c) $O(n^3)$

d) $O(n^5)$

$\rightarrow$ n iteration (i variable)

for $i = n/2$

$j \rightarrow \frac{n}{2} \times \frac{n}{2}$

$j <= i * i$

$k = \frac{n}{2} \times \frac{n}{2}$

$\frac{n}{2} \times \frac{n}{2} \times \frac{n}{2} \times \frac{n}{2} \times \frac{n}{2}$

$\frac{n^5}{2^5} \Rightarrow k \times n^5$

$\rightarrow O(n^5)$

a) $O(n)$
b) $O(n \log n)$
c) $O(n^2)$
d) $O(n^2 \log n)$ ✓

n^2 $O(n)$ for ($i = n/2$; $i <= n$; i++) $\frac{n}{2} \to n$

$O(n)$ for ($j = 1$; $j + \frac{n}{2} \le n$; j++)

$O(\log_2 n)$ for ($k = 1$; $k \le n$; k *= 2)

$k = 1 \to 2 \to 4 \to 8 \dots\dots \to 2^{l}$ l → no of item

$$2^l \le n$$

$$l \le \log_2 n$$

a) O(n)

b) O(n)log n)

c) O(n^2)

d) O(n^2 log n)

for(i=1; i<=n; i++) → O(n)

for(j=1; j<=n; j++) → Cost

break;

O(n)

→ log formula

1) $\log a^b = b \log a$

2) $\log xy = \log x + \log y$

3) $\log x/y = \log x - \log y$

4) $a^{\log_b x} = x^{\log_b a}$

a, b cont

5) $\log_b^2 n = (\log_b n)^2$

b is cont, 2 is cont

6) $a^{\log_a n} \Rightarrow n$

(a is cont)

7) $\log_b x \longrightarrow \dfrac{\log_a x}{\log_a b}$

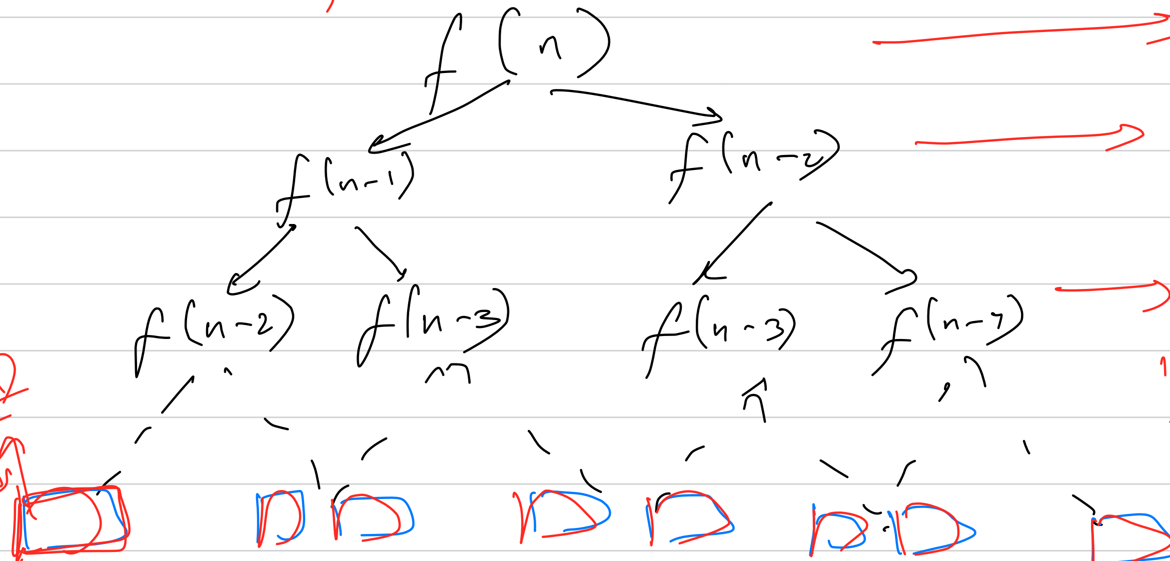
$\boxed{a, b \text{ is cont}}$

In Recursion → Recursion tree

→ fibonacci

Recursion tree

$$f(n) = f(n-1) + f(n-2)$$



$\rightarrow 2^0$

$\rightarrow 2^1$

$\rightarrow 2^2$

$\rightarrow 2^n$

no of nodes × O(1)

$T(n)$ → total operation

1 → const

$$T(n) = 2^n \times 1 + 2^{n-1} \times 1 \text{ - - - - - - - - } 2^2 \times 1 + 2^1 + 2^0$$

$O(2^n)$ → recursive fib

$f(n) = \underbrace{n \times f(n-1)}$

n node

$\overbrace{1+1+1 \cdots\cdots)}$ $\longrightarrow$ $f(n) + O(1)$

$\longrightarrow$ $f(n-1) \to O(1)$

$O(n)$

$\longrightarrow$ $f(n-2) \to O(1)$

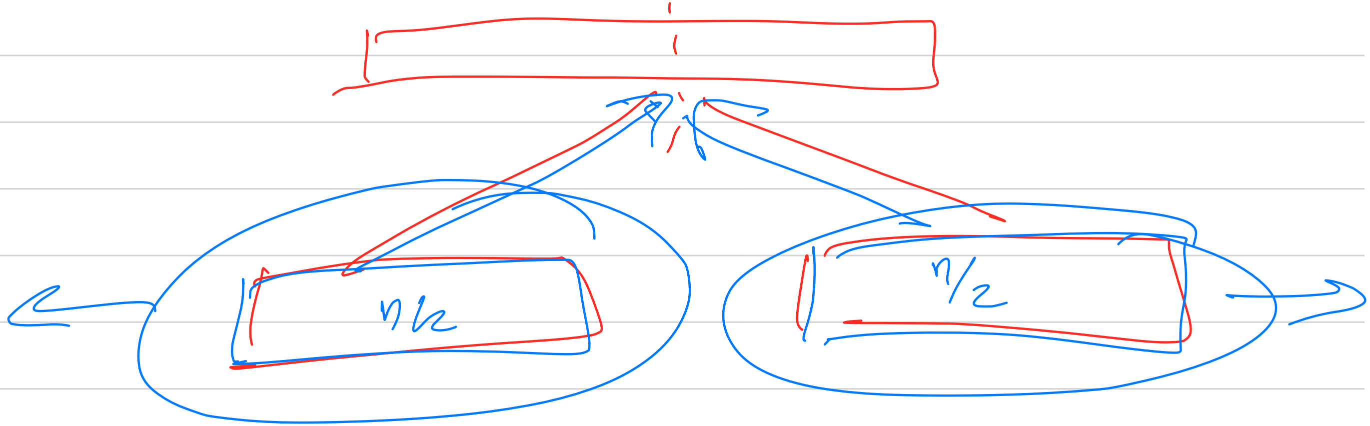
$\longrightarrow$ $f(n-3)$

$\vdots$

$f(1) + \underline{O(1)}$

$$\underbrace{T(n) = 2T\left(\frac{n}{2}\right) + O(n)}$$ → merge sort

$O(n)$ merge



DNC

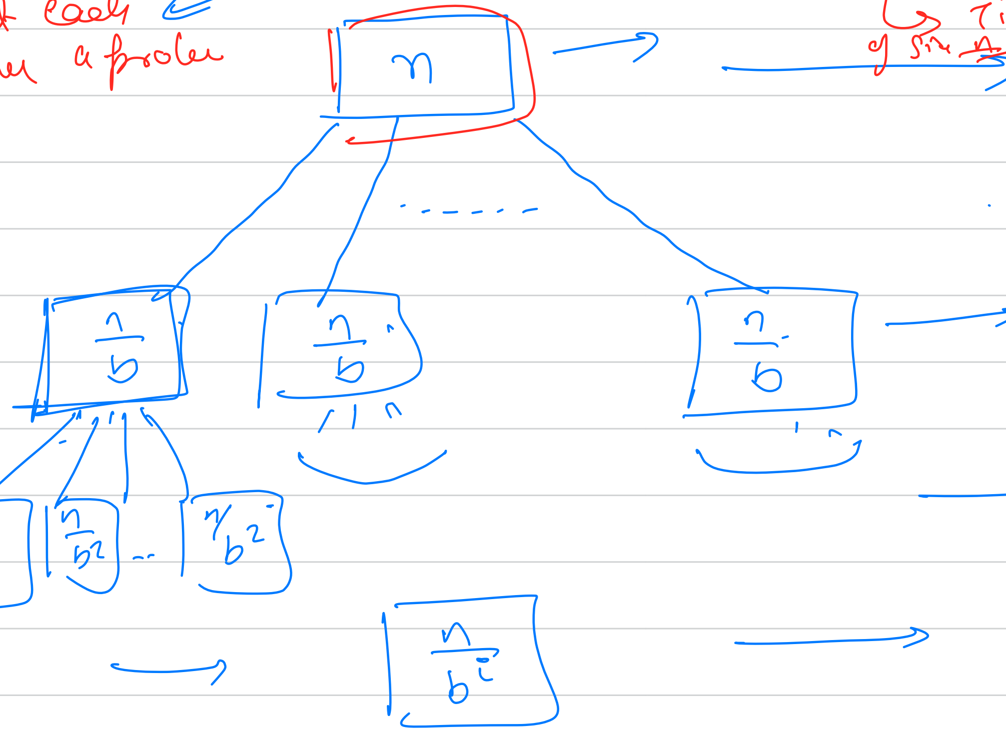
general eqn

$$T(n) = a \times T\left(\frac{n}{b}\right) + O(n^d \log^p n)$$

size

at each level a problem

Time to solve 1 prob of size n



1 problem

a instances of $\frac{n}{b}$ size prob

a problem

a^2 proble

a^i problems

$n \to \frac{n}{b} \to \frac{n}{b^2} \to \frac{n}{b^3} - - - - - \frac{n}{b^k}$

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

$k = \log_b n$

n

$\frac{n}{b} \quad \frac{n}{b} \quad - - - - \quad \frac{n}{b}$

$\frac{n}{b^2} \quad \frac{n}{b^2} \quad - - - \quad \frac{n}{b^2} \quad \frac{n}{b^2} \quad \frac{n}{b^2} \quad \frac{n}{b^2} \quad \frac{n}{b^2} \quad \frac{n}{b^2} \quad \frac{n}{b^2}$

0 level → $O(n^d)$

1st level → a ↳ $a \times O\left(\frac{n}{b}\right)^d$

2nd level → a^2 problem

ith level → $\frac{n}{b^i}$ → a^i problem

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

level $\log_b n$: $1 \quad 1 \quad 1 \quad - - - - - - \quad 1$ → $O(1) \times a^{\log_b n}$

↳ $O(n^{\log_b a})$

leaves

Total time $\Rightarrow$ $n^d + a \times \left(\frac{n}{b}\right)^d + \cdots \left(\frac{a}{b^i}\right)^d \times n^d + \cdots$

$n^{\log_b a}$

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

$$a + \frac{a}{2} + \frac{a}{2^2} \cdots$$

$$a + ar + ar^2 - - - - - - - ar^{n-1}$$

Sum of gp $\rightarrow a \times \frac{(1-r^n)}{1-r} \Rightarrow$

$O(a)$ — $r < 1$

$O(a \times r^{n-1})$ — $r > 1$

no. of term $\times a \rightarrow r = 1$

$$\sum_{i=0}^{i=\log_b n} O(n^d) \times \left(\frac{a}{b^d}\right)^i \approx a r^i \qquad a = O(n^d)$$

$$\rightarrow r = \left(\frac{a}{b^d}\right)$$

Case I $\quad \frac{a}{b^d} < 1 \rightarrow a < b^d \rightarrow \log_b a < d$

$\hookrightarrow O(n^d)$

Case II $\quad \frac{a}{b^d} = 1 \rightarrow \overbrace{\left(1 + \log_b n\right)}^{\text{no of lev}} \times O(n^d)$

a

$a = b^d$

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

$d = \log_b a$

$\hookrightarrow O(n^d \log n) \rightarrow O(n^{\log_b a} \times \log n)$

Case III

$a > b^d$

$\frac{a}{b^d} > 1 \Rightarrow d < \log_b a$

$\rightarrow O\left(\underline{(n^d)} \times \left(\frac{a}{b^d}\right)^{\log_b n} \right)$

$\xrightarrow{} a$ $\quad \gamma$ $\quad \xrightarrow{} (n-1)$

$O\left(n^d \times \frac{a^{\log_b n}}{b^{d \times \log_b n}} \right) \approx O\left(\cancel{n^d} \times \frac{a^{\log_b n}}{\cancel{n^d}} \right)$

$\approx O\left(a^{\log_b n}\right) \approx O\left(n^{\log_b a}\right)$

$\boxed{b^{\log_b n} \Rightarrow n}$

Master Theorem

$p \to$ real, $a \geq 1$, $b > 1$, $d \geq 0$

$\longrightarrow T(n) = a \times T\left(\frac{n}{b}\right) + O(n^d \log^p n)$

$b \to$ real
no

1. $a < b^d \longrightarrow p \geq 0 \longrightarrow O(n^d \log^p n)$

   $p < 0 \longrightarrow O(n^d)$

2. $a = b^d \longrightarrow p > -1 \longrightarrow O(n^{\log_b a} \times \log^{p+1} n)$

   $p = -1 \longrightarrow O(n^{\log_b a})$

   $p < -1 \longrightarrow O(n^{\log_b a})$

3. $a > b^d \longrightarrow O(n^{\log_b a})$

$\log a^{b}$

$b \log a$

$$n \rightarrow \frac{n}{b} \rightarrow \frac{n}{b^2} \rightarrow \frac{n}{b^3} \text{ - - - - - - - - } \frac{n}{b^k}$$

GP

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

$$k = \log_b n$$

$b = 2$

$$\log n \times (\log n)^{-1}$$

Q ⇒ $T(n) = 3T(n/2) + n^2$

a) $O(n)$

b) $O(n^2)$ ✓

c) $O(n^2 \log_2 3)$

d) None

$a = 3$

$b = 2$

$d = 2$

$p = 0$

$3 < 2^2$

$\underline{a < b^d}$

$O(n^2 (\log n)^0)$

$\Rightarrow \underline{O(n^2)}$

Q) $T(n) = 4T(n/2) + n^2$

a) $O(n \log n)$
b) $O(n^2)$
c) $O(n^2 \log n)$ ✓
d) None

$p = 0$

$a = 4$

$d = 2$

$b = 2$

$2^2 = 4$

$a = b^d$

$O(n^{\log_b a} \times \log^{p+1} n)$

$O(n^{\log_2 4} \times \log^1 n)$

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

Q-9 $T(n) = 2T(n/4) + n^{0.51}$

a) $O(n)$
b) $O(n \log n)$
c) $O(\log n)$
d) None ✓

$a = 2$
$b = 4$
$d = 0.51$
$p = 0$

$a \quad b^d$

$2 < 4^{0.51}$

$\sqrt[0.51]{4} > 1/2$

$O(n^d \times \log^p n)$

$O(n^{0.51})$

Q => $T(n) = 3T(n/3) + n/2$

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

b) $O(\log n)$

c) $O(n \log n)$

d) None

Case 2

Q ⇒ $T(n) = 2^n T(n/4) + n^n$

a) $O(1)$

b) $O(\log n)$

c) $O(n)$

d) $O(n^n)$ ✓

$a = 2^n$

$b = 2^2$

$d = n$

$p = 0$

Q⇒ $T(n) = \begin{cases} 3T(n-1) & \forall n > 0 \\ 1 & \text{otherwise} \end{cases}$

$\Rightarrow T(n) = 3\,T(n-1)$

$= 3\,(3\,T(n-2)) = 3^2\,T(n-2)$

$\Rightarrow 3\,(3\,(3\,T(n-3))) \approx 3^3\,T(n-3)$

$\vdots$

$T(n) = 3^n \times \underbrace{T(n-n)}_{\to 1}$

$T(n) = O(3^n)$

$$T(n) = \begin{cases} 2T(n-1) - 1 & \forall n > 0 \\ 1 & \text{otherwise} \end{cases}$$

$$T(n-1) = 2T(n-2) - 1$$

$$T(n) = 2T(n-1) - 1$$

$$\Rightarrow 2(2T(n-2) - 1) - 1 \Rightarrow 2^2 T(n-2) - 2 - 1$$

$$\Rightarrow 2^2(2T(n-3) - 1) - 2 - 1 \Rightarrow 2^3 T(n-3) - 2^2 - 2^1 - 2^0$$

$$\Rightarrow 2^3(2T(n-4) - 1) - 2^2 - 2^1 - 2^0 \Rightarrow 2^4 T(n-4) - 2^3 - 2^2 - 2^1 - 2^0$$

$$\vdots$$

$$\Rightarrow 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} - \cdots 2^2 - 2^1 - 2^0$$

$$\Rightarrow 2^n - (2^{n-1} + 2^{n-2} + 2^{n-3} - \cdots 2^2 + 2^1 + 2^0)$$

$$\Rightarrow 2^n - \underbrace{(2^{n-1} + 2^{n-2} + 2^{n-3} \cdots\cdots 2^2 + 2^1 + 2^0)}$$

GP $\rightarrow$ $a = 1$

$r = 2$

total $\rightarrow n$

$$\frac{1 \times (2^n - 1)}{2-1} \Rightarrow 2^n - 1$$

$$T(n) \Rightarrow \cancel{2^n} - (\cancel{2^n} - 1)$$

Math OP

$$T(n) = 1$$

$\hookrightarrow$ $O(1)$

Q) $T(n) = \sum_{i=1}^{n} \log i$

$n! \le n^n$

$T(n) = \log 1 + \log 2 + \log 3 \cdots\cdots\cdots + \log n$

$\log x + \log y = \log xy$

$T(n) = \log(1 \times 2 \times 3 \cdots\cdots \times n-1 \times n) = \log(n!)$

$\log(n!) \le \log n^n = n \log n$

$O(n \log n)$

Q→

$T(n) = 2T(\sqrt{n}) + \log n$ → Master theorem

Substitution assume $\boxed{n = 2^m}$ - $m = \log n$

$\Rightarrow 2T(\sqrt{2^m}) + m$

$\Rightarrow 2T(2^{m/2}) + m$

assume → $S(m) = T(2^m)$ ✓

$T(n) \Rightarrow T(2^m) \Rightarrow S(m) = 2 \times S\left(\frac{m}{2}\right) + m$

↳ $O(m \log m)$

↳ $O(\log n \times \log\log n)$ ✓

Q:

$$T(n) = T(n-3) + O(n^2)$$

$$\Rightarrow \left(T(n-6) + n^2\right) + n^2$$

$$\Rightarrow T(n-9) + n^2 + n^2 + n^2$$

⋮

$$T(n-3k) + 3k \times n^2$$

$$\Rightarrow T(n-n) + \cancel{3} \times \frac{n}{\cancel{3}} \times n^2 \rightarrow O(n^3)$$

$$\underbrace{n \rightarrow n-3 \rightarrow n-6 \rightarrow n-9 \text{ - - - -}}_{n/3}$$

$k = n/3$

$n \to 7$

$i = 2$

$n = 36 \longrightarrow$

1×36

2×18

3×12

4×9

6×6

9×4

12×3

18×2

36×1

prime or composite

$2 \longrightarrow 35$

$\sqrt{36}$

$2 - \sqrt{36}$

Q2

$1 \to 2 \to \dots \; n$

```
bool isItSafe(vector<vector<bool> > &grid, int row, int col, int n) {
    // O(n)
    // column check
    for(int i = row-1; i >= 0; i--) {
        if(grid[i][col]) return false;
    }
    // left upper diagnol
    for(int i = row-1,j = col - 1; i>=0 and j>=0;i--, j--) {
        if(grid[i][j]) return false;
    }
    // right upper diagnol
    for(int i = row-1, j=col+1;i>=0 and j<n;i--, j++) {
        if(grid[i][j]) return false;
    }
    return true;
}
```

$3n \to n$

$$T(n) = nT(n-1) + O(n^2)$$

$$T(n) = n \times (n \times T(n-2) + O(n^2)) + O(n^2)$$

$$= n(n(n \times T(n-3) + O(n^2)) + O(n^2 + O...$$

```
void countNQueen(vector<vector<bool> > &grid, int curr_row, int n) {
// base case
    if(curr_row == n) {
        // we found one way
        queenCount++;
        display(grid, n);
        cout<<"\n\n";
        return;
    }

    for(int i = 0; i < n; i++) {
        // i will go to each spot/column
        if(isItSafe(grid, curr_row, i, n)) { // to be implemented
            grid[curr_row][i] = true;
            countNQueen(grid, curr_row+1, n);
            grid[curr_row][i] = false; // reinitialisation
        }
    }
}
```

$1 \to 2 \quad n \to 3$

$$T(n) = n \times T(1) + n \times O(n^2)$$

$$O(n^n)$$

Space Complexity

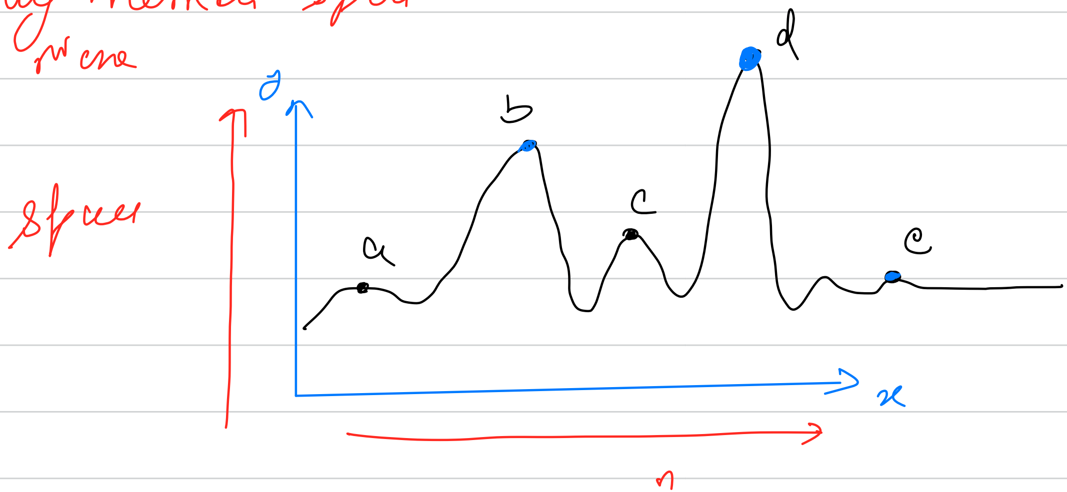
→ max space taken by algo at any point of time during execution

→ a) total memory space

→ b) max memory space

→ c) avg memory space

→ d) none



2gbram

Spacetime tradeoff → This rule says that, sometime for some algo, in order to reduce the time of execution we might spend extra space, or sometimes to save the space exhausted we might increase the time of execution.

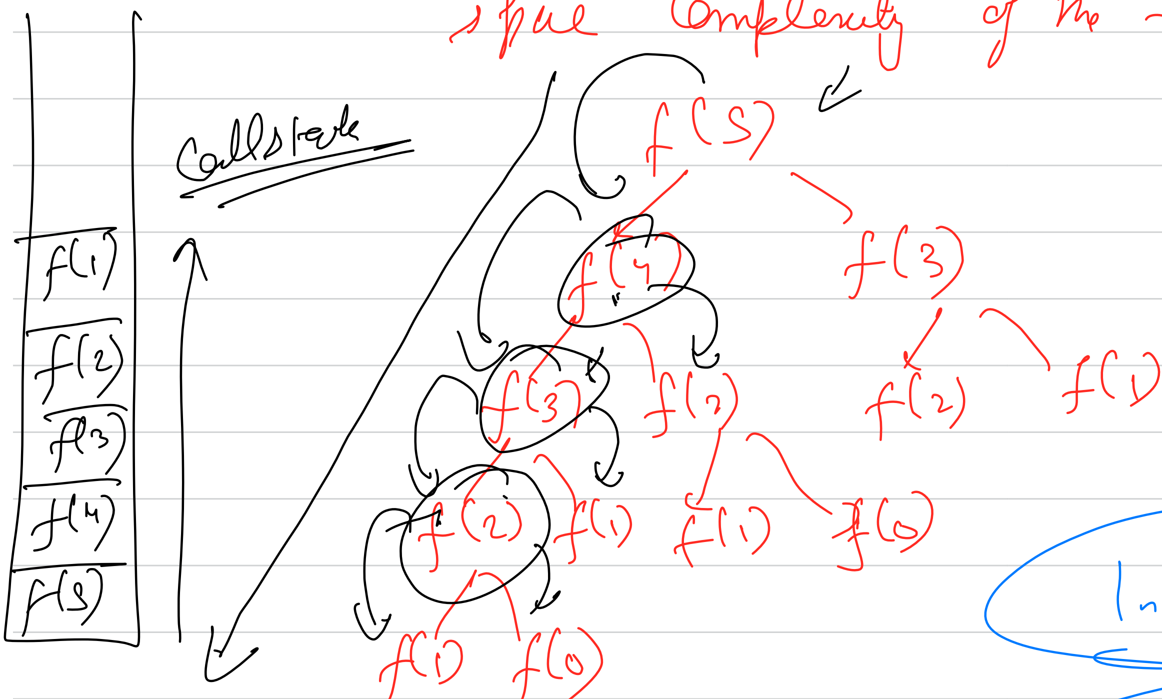
Merge Sort → $T.C \rightarrow O(n \log n)$ + $O(n)$

Inplace merge Sort → $O(n^2)$

Fib → DP

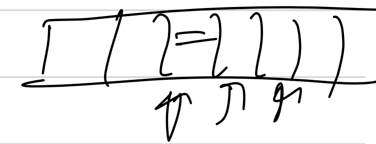
$$f(n) = \underbrace{f(n-1) + f(n-2)}$$

space complexity of the recursive sol^n



$O(n)$

TD DP



$O(n)$

Interviewbit