TIME COMPLEXITY

TC of an algo is the time taken by an algo to run as a function of "LENGTH OF INPUT".

This is NOT same as execution time.

WORST-CASE Scenario ⇒ Big-O-Notation.

order of growth depends on the length of the INPUT.

SPACE-COMPLEXITY: The amt of memory required by algorithm to solve & is it called SC.

Fixed-part: Independent of input size → Codes, Const, Var.
Variable-part: Dependent on input size
↳ Memory for Recession Stacks

Auxillary space complexity: Extra space used in a algo. apart from user I/P.

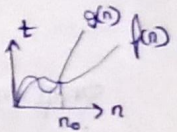
ASYMPTOTIC NOTATION

It is a way to describe TC, SC, by analysing the input size. time (space) is calculated with given input size.

3 NOTATIONS ⇒ Big-O, Omega (Ω), Theta (Θ).

Big-O:

$f(n)$ → actual TC
$g(n)$ → estimated TC.



UPPER BOUND
worst case scenarion - maxt

$f(n) \leq c \cdot g(n) \Rightarrow f(n) = O(g(n)), \quad n \geq n_0$.

n_0 - BREAK POINT EVENT.

Eg①: $f(n) = n$, $g(n) = 2n$

WKT $f(n) \leq c \cdot g(n)$

$n \leq c \cdot 2n$

For $c = 1$, $n = 1$

$1 < 2$

$f(n) = O(n)$.

Eg②: $f(n) = 4n + 15$, $g(n) = n$

WKT $f(n) \leq c \cdot g(n)$

$4n + 15 \leq c \cdot n$

For $c = 5$,

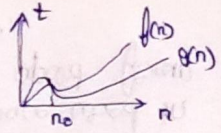
$4n + 15 \leq 5n$

$15 \leq n$

$n_0 = 15$

Omega (Ω):

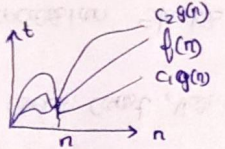
LOWER-BOUND
Best Case Scenario. -mint.



$f(n) \geq c \cdot g(n) \Rightarrow f(n) = \Omega(g(n))$, $n \geq n_0$

Theta (Θ):

Average Case TC.



$c_1 g(n) \leq f(n) \leq c_2 g(n)$, $\exists c_1, c_2 \in R$, $n_0 \geq n$

$f(n) \equiv T(n)$

① SUBSTITUTION METHOD:

Back Substitution.

① $T(n) = T(n-1) + 1$

$= T(n-2) + 2$ $\quad T(n-1) = T(n-2) + 1$

$= T(n-3) + 3$ $\quad T(n-2) = T(n-3) + 1$

⋮ K times

$= T(n-K) + K$.

$T(n) = T(n-K) + K$.

From q.

$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$

when $n = K \Rightarrow T(n) = T(n-n) + n = n$

$T(n) = n$.

```
Void test(int n){
    if(n>0){
        printf("%d", n);
        test(n-1);
    }
}
```

② $T(n) = T(n-1) + n$

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

$= T(n-3) + (n-2) + (n-1) + n$, $T(n-2) = T(n-3) + n-2$

$= T(n-4) + (n-3) + (n-2) + (n-1) + n$

⋮ K times.

$= T(n-K) + (n-(K-1)) + (n-(K-2)) + \dots \; n-1 + n$

When $n = K$, $T(0) = 1$

$T(n) = 0 + 1 + 2 + \dots + n-1 + n$

$= \frac{n(n+1)}{2} = n^2 \Rightarrow T(n) = O(n^2)$.

This is back substitution, since we are finding the prev terms of each term. Like $n-1, n-2, \dots$

Log

$y = b^x$

$\log_b y = x \cdot \log_b b = x$

$\log_b y = x$, $y > 0$, $\log y > 0$, $b > 1$

Log is inverse functions on exponents.

③ $T(n) = T(n-1) + \log n$

$= T(n-2) + \log(n-1) + \log n$

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

⋮

$= T(n-K) + \log(n-(K-1)) + \dots + \log n$

when $n = K \Rightarrow T(n) = 0 + \log 1 + \dots + \log n$

$= \log(1 \cdots n) = \log n! = n \log n$

④ $T(n) = 2T(n-1) + 1$

$T(n) = 2 \times 2T(n-2) + 2 + 1 = 4T(n-2) + 2 + 1$

$= 8T(n-3) + 4 + 2 + 1$

⋮ K times

$= 2^K T(n-K) + \underbrace{2^{K-1} + 2^{K-2} + \cdots + 2 + 1}$

This follows a GP

$a = 1, r = 2$

Wkt $\Rightarrow a + ar + ar^2 + ar^3 + \cdots + ar^K = \frac{a(r^{K+1}-1)}{(r-1)}$

$\therefore T(n) = 2^K T(n-K) + 1 \cdot \frac{(2^{K+1}-1)}{2-1}$

$= 2^K T(n-K) + \frac{1 \cdot 2^K}{K-2}$

When $n = K \Rightarrow T(n) = 2^K$.

CONCLUSIONS:

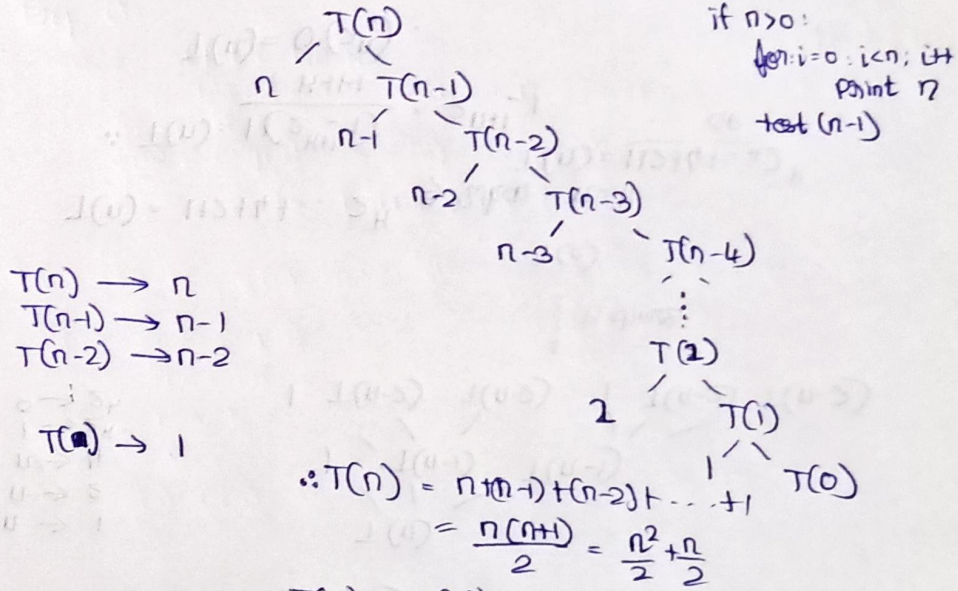
$T(n) = T(n-1) + 1$	$O(n)$
$T(n) = T(n-1) + n$	$O(n^2)$
$T(n) = T(n-1) + \log n$	$O(n \log n)$
$T(n) = T(n-1) + n^2$	$O(n^3)$
$T(n) = T(n-2) + 1$	$O(n/2) \simeq O(n)$
$T(n) = 2T(n-1) + 1$	$O(2^n)$
$T(n) = 2(n-1) + \log n$	$O(2^n \log n)$

② RECURSIVE TREE METHOD

Form a recursive tree with the given recursive relation.

Then find the no of times each function have called out the statments / code of instruction.

① $T(n) = T(n-1) + n$




```
Void test(int n)
  if n>0:
    for:i=0 ; i<n; i++
      print n
    test(n-1)
```

$T(n) \longrightarrow n$

$T(n-1) \longrightarrow n-1$

$T(n-2) \longrightarrow n-2$

⋮

$T(1) \rightarrow 1$

$\therefore T(n) = n + (n-1) + (n-2) + \cdots + 1$

$= \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

$T(n) = O(n^2)$

② $T(n) = T(n-1) + \log(n)$

$$T(n) = \begin{cases} 1 & , n=0 \\ T(n-1) + \log n & , n>0 \end{cases}$$

```
Void test(n)
if n>0:
    for(i=1; i<n; i=i*2
        print n
    test(n-1)
```

$i = i*2 \Rightarrow$ 1 2 4 8 16 32 64 ..... 2^{k-1}

$n = 2^k$

How log? ↓

$\log_2 n = k$

$T(n)$

$\log n$ | $T(n-1)$

$\log(n-1)$ | $T(n-2)$

$\vdots$

$T(2)$

$\log_2$ | $T(1)$

$\log_1$ | $T(0)$

$T(n) = \log 1 + \log 2 + \ldots \log(n-1) + \log n$

$= n \log n = \log n!$

③ Algo

```
test(int n)
if n>0
    print n
    test(n-1)
    test(n-1)
```

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

$n \to 1$

$n^1 \to 2$

$n^2 \to 4$

$1 \to 2^{k-1}$

$0 \to 2^k$

$T(n)$

1 | $T(n-1)$ | $T(n-1)$

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

$\vdots$ k times

$T(0)$.

$T(n) = 1+2+4+\ldots 2^k \Rightarrow$ when $n=k$

$T(n) = \underbrace{1+2+4+\ldots+2^k}_{GP}$

$\therefore T(n) = \frac{1(2^{k+1}-1)}{k+1-1} = 2^{n+1}-1$

$T(n) = O(2^n)$.

DIVISION FUNCTIONS

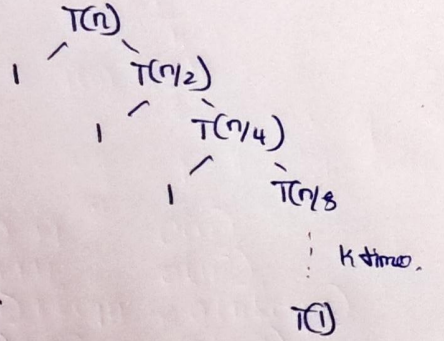
① $T(n) = T(n/2) + 1 \quad n > 1$

$T(n) = T(n/2) + 1$
$= T(n/4) + 2$
$= T(n/8) + 3$
$\vdots \times k$
$= T(n/2^k) + k$

$T(n) = T(n/2^k) + k$

wkt @ the end $\frac{n}{2^k} = 1$, $n = 2^k$ } $k = \log_2 n$

$\therefore T(n) = T(1) + \log_2 n$
$T(n) = \log_2 n$



$T(n) = 1 \cdot k = k$.
$= \log_2 n$.

Algo test(int n)
if (n>1)
print n
test(n/2)

$\therefore T(n) = 1 \quad n = 1$

② $T(n) = T(n/2) + n$

$T(n/2) = T(n/4) + \frac{n}{2} \Rightarrow T(n) = T(n/4) + n + \frac{n}{2}$
$T(n/4) = T(n/8) + \frac{n}{4} \Rightarrow T(n) = T(n/8) + n + \frac{n}{2} + \frac{n}{4}$

k times

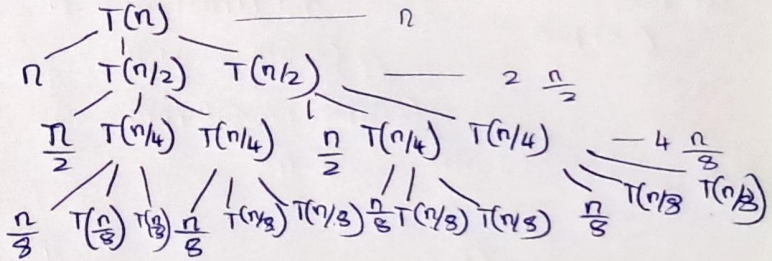
$= T(n/2^k) + n + \underbrace{\frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^k}}_{GP}$

wkt $\frac{n}{2^k} = 1$
$k = \log_2 n$.

$\therefore T(n) = T(0) + n\left(\frac{1}{2} + \frac{1}{4} + \dots \frac{1}{2^k}\right)$ → Lets consider for 1
$= T(0) + \frac{n \cdot 1/2}{1 - \frac{1}{2}} = n$

$\therefore T(n) = O(n)$

③ $T(n) = 2T(n/2) + n$.



$T(1) = T(n/2^k)$

$T(n) = n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{8} + 8 \cdot \frac{n}{16} + \dots + 2^{k-1} \cdot \frac{n}{2^k} + 2^k T(1)$

$= 2^k T\left(\frac{n}{2^k}\right) + nk \quad n\left(1 + 1 + \frac{1}{2} + \frac{1}{2} + \dots \frac{1}{2^k}\right)$

$= 2n + n\left(\frac{k}{2}\right) = 2n + \frac{n}{2} \cdot \log_2 n$

$T(n) = O(n \cdot \log n)$