Dated:

Algorithm vs	Program.
1- Design	1- Implementation
2- Domain knowledge	2- Programmer
3- Any language	3- Programming lang.
4- H/w & OS	4- H/w & OS.
5- Analyze	5- Testing.

Important points for algorithms
- Time
- Space
- Network consumption rate / data transfer rate.
- Power consumption.
- CPU register // (Algorithm for drivers)

- Priori Analysis
  - Algorithm
  - Hardware independent
  - Independent of language.
  - Time and space function.

- Posteriori Analysis/Testing
  - Program.
  - Hardware dependent.
  - Language dependent.
  - Watch time & bytes (mem allocation)

1.2 Characteristics of an algorithm.

1- Input ⟶ 0 or more.
2- Output ⟶ Atleast 1.
3- Definiteness ⟶ shouldn't be unclear/invalid i.e $\sqrt{-1}$ is maths error.
4- Finiteness. ⟶ must terminate at a point
5- Effectiveness.

Dated: **Complexities**

```
swap(a,b) {
    temp = a        ---- 1      a — 1
    a = b           ---- 1      b — 1
    b = temp        ---- 1      temp — 1
}                   f(n) = 3    S(n) = 3
                    O(1)        O(1)
```

Frequency count algorithm

```
sum(A,n) {                          time        space
    s = 0                           ---- 1
    for (i=0 ; i<n ; i++)           ---- (n+1)  s — 1
    {                                           i — 1
        s = s + A[i]                ---- n      A — n
    }                                           n — 1.
    return s ;                      ---- 1      S(n) = n + 3
}                                   f(n) = 2n + 3   O(n)
                                    O(n)
```

```
MatrixAdd(A,B,n)                                    Space
{
 for(i=0 ; i<n ; i++)               ---- n+1        A,B,C — 3(n²)
 {                                                  n — 1
  for(j=0 ; j<n ; j++)              ---- n(n+1)     i — 1
  {                                                 j — 1
n   c[i][j] = A[i][j] + B[i][j]     ---- n x n      S(n) = 3n² + 3.
  }                                                 O(n²).
 }                                  f(n) = 2n² + 2n + 1
}                                   O(n²)
```

Dated:

```
MatrixMultiply(A,B,n)
{
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
    {
      C[i,j] = 0
      for(k=0;k<n;k++)
      {
        C[i,j] = C[i,j] + A[i,k] * B[k,j]
      }
    }
  }
}
```

time	space
n+1	A,B,C → $2n^2$
n^2+n	i → 1
n^3	j → 1
n^3+n^2	n → 1
n^3	k → 1
$f(n)=2n^3+3n^2+2n+1$	$S(n)=2n^2+4$
$O(n^3)$	$O(n^2)$

same result

① `for(i=0;i<n;i++)` — $n+1$
`{ ——— }` — n
$2n+1 \Rightarrow O(n)$

② `for(i=0;i<n;i+=2)` → if we change it to any no. it will remain
`{ ——— }` — $n/2$
$O(n)$

③ `for(i=0;i<n;i++)` — $n+1$
`for(j=0;j<n;j++)` — $n(n+1)$
`{ ——— }` — $n \times n$
$O(n^2)$

④ `for(i=0;i<n;i++)` — $n+1$ (j is increasing according to i)
`{ for(j=0;j<i;j++)` — $1+2+3\cdots n = \frac{n(n+1)}{2}$
`{ ——— }`
`}`
$f(n) = \frac{n^2+n}{2} \Rightarrow O(n^2)$

⑤ `P = 0`
`for(i=1;p<=n;i++)` — $O(\sqrt{n})$
`{ P = P + i }`

i	P
1	0+1
2	0+1+2 = 3
⋮	⋮
k	0+1+2+3⋯k

∵ it will stop when $p > n$.
∴ $p = \frac{k(k+1)}{2}$
$\frac{k(k+1)}{2} > n \Rightarrow \frac{k^2+1}{2} > n$
$\Rightarrow k^2 > n \Rightarrow k > \sqrt{n}$

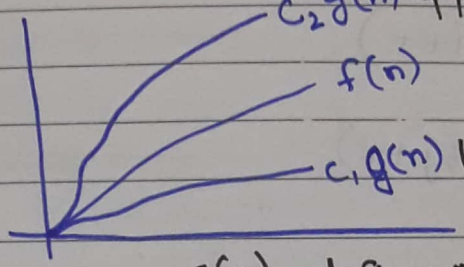
Dated:

negation price → -ve amount of time (remaining time).

theta notation.

↙

tell the upper & lower bound of f(n) any function.



g(n) defines the boundaries of f(n).

C_1 =

C_2 =

① $f(n) = 10n^3 + 5n^2 + 17$.

Best case : when $5n^2 + 17 = 0$.

$= 10n^3$ → $C_1 = 10$

Worst Case when each term has n^3.

$= 10n^3 + 5n^3 + 17n^3 = 42n^3$

$C_2 = 42$

② $f(n) = 10n^3 + n\log n$.

negligible as it will be ≤ 1

$0 \rightarrow 1$

Best

$f(n) = 10n^3 \rightarrow C_1 = 10$

worst :

$10n^3 + 1n^3 \log(n^3)$

$= 10n^3 + 1n^3$

$= 11n^3 \rightarrow C_2 = 11$.

Dated:

recurrance relation ⟶ recursively defined function.

$\log_2(32) = 5$ as $2^5 = 32$.

```
T(n)
{
  if (n > 1)
    print (n)        ——— 1
  T(n-1)   ⟶   T(n-1)
}
```

Methods to solve Recursion

Iterative method

Recurrance tree

Master theorem

$$T(n) = 2T(n/2) + n.$$

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

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

$$= 8\left(2T\frac{n}{16} + \frac{n}{8}\right) - - - -$$

4/Sep/19

Proof Using loop Invariant ⟵ Initialization. / Maintenance. / Termination. } example : insertion sort

Dated:

1.5.2 time complexity.

```
for (int i=1 ; i<n ; i=i*2)
{
    ------
}
```

here due to multiplication the time complexity will be in $\log_2 n$

$(\log n)$

it gives a decimal value so we need to take ceil or floor.

i
$1 = 2^0$
$1 \times 2 = 2 = 2^1$
$2 \times 2 = 4 = 2^2$
$2^2 \times 2 = 8 = 2^3$
$2^3 \times 2 = 16 = 2^4$
⋮
$2^k \times 2$ $\quad$ 2^k

$i >= n$

$2^k >= n.$

$2^k = n$

$k = \log_2 n$

$\longrightarrow O(\log_2 n)$

vs

```
for(i=1;i<=n;i++)
{
    ------
}
```

$i = \underbrace{1+1+1+1---1}_{k} = n$

$k = n.$

$O(n).$

```
for (i=1 ; i<n ; i=i*2)
{
    statement
}
```

let n=8

p=0	n=8 i		if n=10 i	
1	1	3	1	4
2	2		2	
3	4		4	
	8 X		8	
			16 X	

$\log 8 = 3$

$\log_2 2^3 = 3 \log 2$

$= 3$

$\log_2 10 = 3.2$

$\lceil \log n \rceil = 4$ (ceil)

$\log_{10} 1000 = 3$.
$10 \times 10 \times 10 = 1000$
$10^3 = 1000$.

Dated:

★ `for (i=n ; i>=1 ; i=i/2)`
`{ statement ; }`

	i
$i < 1$	n
$\frac{n}{2^k} < 1$	$n/2$
$n < 2^k$	$n/2^2$
$n = 2^k$	$n/2^3$
$k = \log_2 n$	⋮
$O(\log_2 n)$	$n/2^k$

★ `for (i=0 ; i*i<n ; i++)`
`{ statement }`

i	
0	0^2
1	$1^2 < k$
⋮	$2^2 < k$
k	⋮

$i^2 >= n$
$i^2 = n$
$i = \sqrt{n} \Rightarrow O(\sqrt{n})$

★ `for (i=0; i<n; i++) { —— }` —— n
`for (j=0; j<n; j++) { —— }` —— n
$2n$
$O(n)$

★ `p=0`
`for (i=1; i<n ; i*=2) { p++; }` $O(\log_2 n)$ $= \log n = P$
`for (j=1 ; j<P ; j=j*2) { statement }` $(\log P)$

i	
1	$= 2^0$
1×2	$= 2^1$
2×2	$= 2^2$
⋮	⋮
	2^k

$j >= P$ $\log(\log n)$
$2^k >= P$
$O(\log_2 P)$

$P = \log n$.
2nd loop
$\Rightarrow \log P$
$\Rightarrow \log(\log n)$
$\Rightarrow O(\log \log n)$. Ans

NOTE

① $y = \log_a x \Rightarrow a^y = x$
② $\log_x(a.b) \Rightarrow \log_x(a) + \log_x(b)$.
③ $b \log(a) = \log a^b$
④ $\log_a a = 1$
⑤ $\log_a 1 = 0$
⑥ $\log_a(x^n) = n \log_a(x)$
⑦ $\log\left(\frac{a}{b}\right) = \log a - \log b$
⑧ $\ln(x) = \log_e(x)$

Note: $\log_2 n = \frac{\log n}{\log 2} = \log n \times \frac{1}{\log 2} = O(\log n)$

$\therefore \log_k n = O(\log n)$.

Dated:

```
for (i=0; i<n; i++)          ——— n
{
  for (j=1; j<n; j=j*2)      — n (log n)
  {
    ———                      ——— n (log n)
  }                          ———————
}                            2n log n + n
                             ⇒ O(n log n)
```

```
for (i=0; i<n; i++) ⇒ O(n)
for (i=n; i>0; i--) ⇒ O(n)
for (i=0; i<n; i+=2) ⇒ n/2 = O(n)
for (i=1; i<n; i*=2)
        ⇒ O(log2 n)
for (i=1; i<n; i*=3)
        ⇒ O(log3 n)
for (i=n; i>1; i=i/2)
        ⇒ O(log2 n)
```

1.5.3 → 'while' and 'if' complexity.

```
i=0          —— 1
while (i<n)  — n+1
{
  statement  — n
    i++      — n
}            ———
             f(n) = 3n + 2
             O(n)
```

```
i = n
while (i>1)
{
  statement
    i = i/2
}
```

i = n, i/2^1, i/2^2, ⋮, i/2^k

$\frac{n}{2^k} = 1$

$n = 2^k \Rightarrow k = \log_2 n$

$O(\log n)$

```
a=1
while (a<b)
{
  statement
    a=a*2
}
```

a: $1 = 2^0$, $1 \times 2 = 2^1$, $2 \times 2 = 2^2$, ⋮, 2^k → Terminate when $a >= b$.

$2^k >= b$, where $a = 2^k$

$2^k = b$

$k = \log_2 b$ let $b = n$

$k = \log_2 n$

$O(\log_2 n) = O(\log n)$

Dated:

equivalent to:
for(k=1; i=1; k<n; i++)
{ statement;
k=k+i; }

```
i=1
k=1
while (k<n)
{
    statement :
    k=k+i
    i++
}
```

i	k
1	1
2	2
3	2+2 = 4
4	2+2+3 = 7
5	2+2+3+4 = 11
⋮	⋮
m	2+2+3+4+5+---- m = $\frac{m(m+1)}{2}$

only this needs to be 1 so it will approximately become equal to $\frac{m(m+1)}{2}$

Terminates at:

~~k~~ $k >= n$

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

$m^2 = n$

$m = \sqrt{n} \Rightarrow O(\sqrt{n})$ Ans.

```
while(m!=n)
{
    if(m>n)
        m=m-n
    else
        n=n-m
}
```

m	n
6	3
3	3

} 1 time

m	n
5	5

} 0 time

$O(1)$

m	n
16	2 (initial values)
14	2
10	2
8	2
6	2
4	2
2	2

} 7 times.

16/2 times

$O(n)$

min time = $O(1)$

max time = $O(n)$

Dated:

```
Algorithm Test(n)
{
    if (n<5)
        print n          ——— 1
    else
        for (i=0; i<n; i++)
            print i      ——— n
}
```

Best : O(1)
worst : O(n)

Also written as :

```
Test(n)
{ if(n>=5)
    for(i=0;i<n;i++)
    { ——
    }
}
```

1.6 : Classes of function

* O(1) ⟶ constant function
  - f(n) = 1
  - f(n) = 20
  - f(n) = 5000
  - ⟶ O(1)

* O(n) ⟶ linear function.
  - f(n) = 2n+3
  - f(n) = 500n + n + 2
  - f(n) = $\frac{n}{5000}$ + 6
  - ⟶ O(n)

* O(log n) ⟶ logarithmic.
* $O(n^2)$ ⟶ Quadratic
* $O(n^3)$ ⟶ Cubic.
* $O(2^n)$ ⟶ exponential. ⟶ e.g : $O(n^n)$, $O(3^n)$, $O(10^n)$.

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots\cdots < 2^n < 3^n < \cdots n^n$.

($n^2\log n$, $n^3\log n$)

Dated: 1·8 Asymptotic Notations.

1.8.1

big-Oh	O( )	upper bound.
big-Omega	Ω( )	lower bound.
Theta-notation	Θ( )	Average bound.

Big-Oh

The function $f(n) = O(g(n))$ iff $\exists$ +ve constant c and n_0 such that $f(n) \leq c * g(n)$ $\forall$ $n \geq n_0$.

i.e $f(n) = 2n+3$

$2n+3 \leq 2n+3n$ (make all terms on R.H.S multiple of n)

$2n+3 \leq 5n$ where $n \geq 1$

$f(n)$ c $g(n)$

$f(n) = O(n)$ ✓

can this function be $O(n^2)$?

Ans: Yes, as: here g(n) is taken as n^2 so,

$f(n) = 2n+3$

$2n+3 \leq 2n^2+3n^2$

$2n+3 \leq 5n^2$ so, $f(n) = O(n^2)$ ✓

$f(n) = O(2^n)$ ✓

$f(n) = O(\log n)$ ✓

$1 < \cdots \sqrt{n} < n < n\log n < \cdots\cdots < n^n$

lower bound — avg case — upper bound.

Big-Omega

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

such that $f(n) \geq c * g(n)$. $n \geq n_0$.

i.e $2n+3 = f(n)$

$2n+3 \geq 1n$ $\forall$ $n \geq 1$ $\longrightarrow$ $f(n) = \Omega(n)$ ✓

$f(n)$ c $g(n)$

$f(n) = \Omega(\log n)$ ✓

$f(n) = \Omega(n^2)$ ✗

we take lower → best case
but they are not same

Dated:

Theta - notation

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

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

e.g. :

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

$$\underbrace{1}_{c_1} \times n \leq \underbrace{2n+3}_{f(n)} \leq \underbrace{5}_{c_2} \times n \longrightarrow f(n) = \Theta(n)$$

Example 2 :

$$f(n) = 2n^2 + 3n + 4$$

$$1n^2 \leq 2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$$

$$1n^2 \leq 2n^2 + 3n + 4 \leq \underbrace{9n^2}_{c_2}$$

$\Omega(n^2)$ (lower bound) — c_1 — $O(n^2)$ (upper bound) — $\Theta(n^2)$.

Example 3 :

$$f(n) = n^2 \log n + n$$

$$1n^2 \log n \leq n^2 \log n + n \leq 10 n^2 \log n$$

$\Omega(n^2 \log n)$ — $\Theta(n^2 \log n)$ — $O(n^2 \log n)$

Example 4 :

$$\therefore f(n) = n! = (n \times (n-1) \times (n-2) \cdots \times 3 \times 2 \times 1)$$

$$= 1 \times 2 \times 3 \cdots \times n$$

→ here comes the use of upper and lower bound as here these two bound will help us $\Theta()$ is not possible!

$$1 < 1 \times 2 \times 3 \times \cdots \times n < n * n \times n \times n \cdots n$$

$$1 \leq n! \leq n^n$$

$\Omega(1)$ — $O(n^n)$

↖ Ω is not possible since we cannot find upper bound as n can be smaller or it can be greater

Dated:

Example 5 :

$f(n) = \log n!$

$\log(1 \times 1 \times \cdots 1) \leq \log(1 \times 2 \times 3 \times \cdots \times n) \leq \log(n \times n \times \cdots n)$

$1 \leq \log n! \leq \log n^n$

$1 \leq \log n! \leq n \log n$

$O(1)$ ↑ $O(n \log n)$

$\Theta()$ is not possible because of unpredictable value of n. Here it may be $\Theta(n^2)$

1.9 Properties of Asymptotic Notations.

★ General Property :

if $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$ → it is true for Ω and Θ also.

e.g :

$f(n) = 2n^2 + 5$ is $O(n^2)$

then

$7 \cdot f(n) = 7 \cdot (2n^2 + 5)$

$= 14n^2 + 35$ is also $O(n^2)$

★ Reflexive Property : → function is an upper bound of itself

if $f(n)$ is given then $f(n)$ is $O(f(n))$

e.g

$f(n) = n^2 \rightarrow O(n^2)$

★ Transitive Property :

if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then, $f(n)$ is O

e.g

$f(n) = n \quad g(n) = n^2 \quad h(n) = n^3$

n is $O(n^2)$ n^2 is $O(n^3)$ n is $O(n3)$

Dated:

* Symmetric Property : → it's only true for Θ notation

if $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$

e.g :

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

$f(n) = \Theta(n^2)$

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

Transpose Symmetric → true for Ω and O.

if $f(n) = O(g(n))$ then $g(n)$ is $\Omega(f(n))$

e.g :

$f(n) = n$ $\quad$ $g(n) = n^2$

n is $O(n^2)$

n^2 is $\Omega(n)$

Other properties :

* if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$\Rightarrow$ $g(n) \leq f(n) \leq g(n)$ → both $g(n)$ are same.

here, $\therefore f(n) = \Theta(g(n))$

* if $f(n) = O(g(n))$ and $d(n) = O(e(n))$

then, $f(n) + d(n) = O(\max(g(n), e(n)))$

e.g: $f(n) = n^2 = O(n^2)$

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

then $O(n^2)$

Dated:

if $f(n) = O(g(n))$
and $d(n) = O(e(n))$
then,
$f(n) * d(n) = O(g(n) * e(n))$

i.e $f(n) = n$, $d(n) = n^2$ $\Rightarrow O(n^3)$

Note
- $a^{\log_c b} = b^{\log_c a}$
- $\log(ab) = \log(a) + \log(b)$
- $\log a/b = \log a - \log b$.
- $\log a^b = b \log a$

1.10.1 Comparison of functions.

$\frac{n}{2}$	$\frac{n^2}{4}$	$\frac{n^3}{8}$
3	9	27
4	16	64

Method 1 : check by putting values.

n^2	n^3
$\log n^2$	$\log n^3$
$2 \log n$	$3 \log n$.

$\therefore 2(\log n) < 3(\log n)$

Method 2 : apply log on b.s

e.g :

$f(n) = n^2 \log n$
$= \log(n^2 \log n)$
$= \log n^2 + \log(\log n)$
$= \log n^2 + \log(\log(n))$
$= 2 \log n + \log(\log(n))$
very small

$g(n) = n(\log n)^{10}$
$= \log(n(\log n)^{10})$
$= \log n + \log(\log n)^{10}$
$= \log n + 10 \log(\log n)$
$= \log n + 10 \log(\log n)$
very small as log already small and $\log(\log n)$ become small

so $f(n) > g(n)$

Dated:

e.g ② :

$f(n) = 3n^{\sqrt{n}}$

$= (3(n)^{\sqrt{n}})$

$= 3\, n^{\sqrt{n}}$

$g(n) = 2^{\sqrt{n}\log_2 n}$

$= 2^{\sqrt{n}\log_2 n}$

$= 2^{\log_2 n^{\sqrt{n}}}$

$= ((n^{\sqrt{n}})^{\log_2 2})$

$= n^{\sqrt{n}(1)}$

$= n^{\sqrt{n}}$

$\therefore f(n) > g(n)$ → but asymptotically, they are equal.

e.g ③

$f(n) = n^{\log n}$

$= \log n^{(\log n)}$

$= \log(n) \times \log(n)$

$= \log^2 n$

apply log again ⇒ $2 \log(\log n)$

$g(n) = 2^{\sqrt{n}}$

$= \log 2^{\sqrt{n}}$

$= \sqrt{n} \log_2 2$

$= \sqrt{n}$

$= \frac{1}{2} \log n$.

$g(n) > f(n)$

eg ④

$f(n) = 2^{\log n}$

$= \log 2^{\log n}$

$= \log n (\log_2 2)$

$= \log n$

$g(n) = n^{\sqrt{n}}$

$= \sqrt{n} \log n$.

so $g(n) > f(n)$.

Dated:

e.g ⑤

$f(n) = 2^n$ $\quad g(n) = 2^{2n}$

$= \log 2^n$ $\quad = \log 2^{2n}$

$= n \log_2 2$ $\quad = 2n \log_2 2$

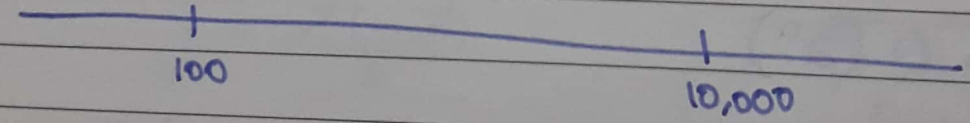
$= n$ $\quad = 2n.$

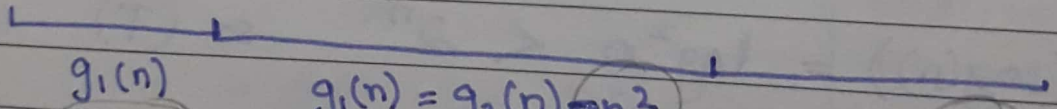
$g(n) > f(n)$ ⟶ but asymptotically they are equal.

eg ⑥

$$g_1(n) = \begin{cases} n^3 & n < 100 \\ n^2 & n \geq 100 \end{cases}$$

$$g_2(n) = \begin{cases} n^2 & n < 10,000 \\ n^3 & n \geq 10,000. \end{cases}$$





$g_1(n)$ $\quad g_1(n) = g_2(n) = n^2$ $\quad g_2(n) = n^3$

$= n^3$

$g_1 > g_2$ $\quad g_2 > g_1$

∴ from 10,000 to ∞ $g_2 > g_1$ so g_2 is grea

Dated:

1.10.2 . T/F

1- $(n+k)^m = \Theta(n^m)$ (T) $\quad \Theta = n^m + k^m + c \rightarrow \Theta(n^m)$ $\quad O = O((1+k)n)^m$

2- $2^{n+1} = O(2^n)$ (T) $\quad 2.(2^n) \rightarrow$ greater.

3- $2^{2n} = O(2^n)$ $\quad 2^2 . 2^n$ X $\quad 4^n > 2^n$

$\rightarrow 4^n$ must have an upper bound greater than 4^n and here $2^n \not> 4^n$

(F)

4- $\sqrt{\log(n)} = O(\log \log n)$ $\quad \sqrt{n} = O(\log n)$

so $\sqrt{n} > \log n$.

$\log\log(n) \not> \sqrt{\log(n)}$ so (F)

5- $n^{\log n} = O(2^n)$

$\log n(\log(n)) = \log^2 n < 2^n$ so (T)

Dated:

1.11 Best, worst and Average case Analysis.

① Linear Search.

A	8	6	12	5	9	7	4	3	16	18
	0	1	2	3	4	5	6	7	8	9

key = 7. → Traverse till index 5

key = 20 → Traverse whole array → $O(n) = W(n)$

Best case → key element is found at first index ⇒ $O(1)$

$B(n) = O(1)$

Average case = $\frac{\text{all possible cases}}{\text{no. of cases.}} = \frac{1+2+3+\cdots+n}{n}$

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

$= \frac{n+1}{2} \Rightarrow O\left(\frac{n+1}{2}\right)$

Best case:

$B(n) = 1$

$B(n) = O(1)$

$B(n) = \Omega(1)$

$B(n) = \Theta(1)$

Worst case:

$w(n) = n$

$w(n) = O(n)$

$w(n) = \Omega(n)$

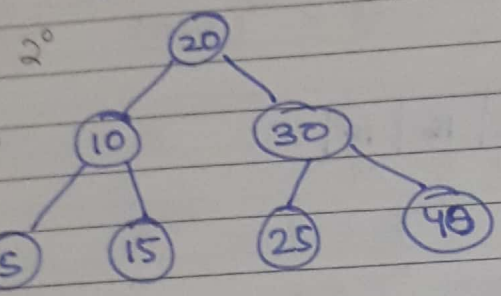
$w(n) = \Theta(n)$

Note:

The best, worst or average case can be represented in any kind of notation. Best case shouldn't be confused with $\Omega()$ and same goes for other notations.

Dated:

Binary search Tree (BST)



Best case → search root element

$B(n) = 1 = O(1)$

Worst case → searching leaf element

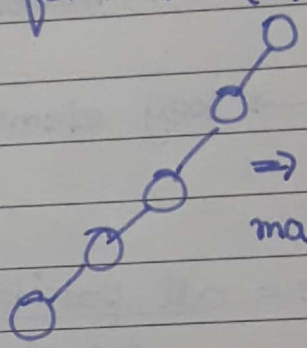
$W(n) = \log n = O(\log n)$ → or $O(\text{height})$ min $W(n)$

nodes at height of tree $= 2^k$

$n = 2^k$

$k = \log_2 n = O(\log n)$ ← height

for max $W(n)$:



⇒ max worst case

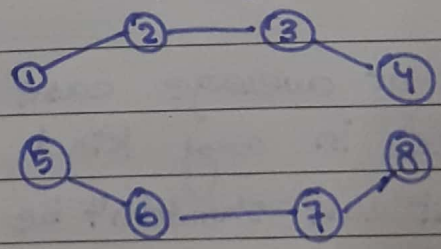
max $W(n) = O(n)$

1.12 : Disjoint Sets

1- Disjoint set & operation.
2- Detecting a cycle.
3- Graphical Representation
4- Array Representation.
5- Weighted Union and collapsing find.

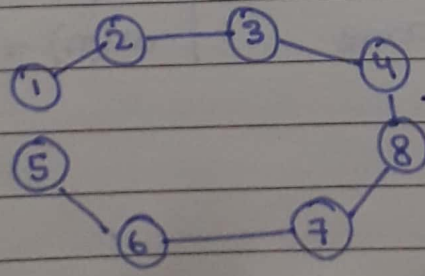
$S_1 = \{1,2,3,4\}$

$S_2 = \{5,6,7,8\}$



→ $S_1 \cap S_2 = \emptyset$

Union :



→ when we perform union we add an edge (4,8). → they both are in diff sets so we perform union

S_1 S_2

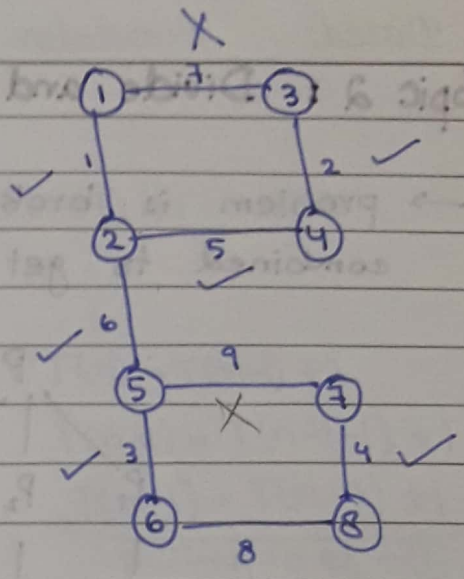
so,

$S_1 \cup S_2 = \{1,2,3,4,5,6,7,8\}$

Dated:

Cycle detection using disjoint sets !

$M = \{1,2,3,4,5,6,7,8\}$



$S_1 = \{1,2\}$

$S_2 = \{3,4\}$

$S_3 = \{5,6\}$

$S_4 = \{7,8\}$

} form sets and cancel them from universal set.

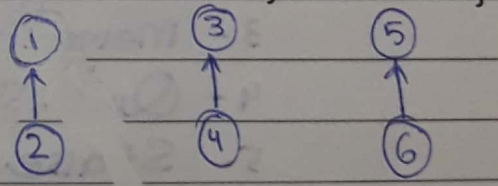
now remaining edges .

★ (2, 4) → in S_1 and S_2 so taking union of both.

$S_1 \cup S_2 = \{1,2,3,4\} = S_5$

★ (1, 3) → both elements are in S_5 so reject it as it's forming a cycle.

$S_1 = \{1,2\}$ $S_2 = \{3,4\}$ $S_3 = \{5,6\}$



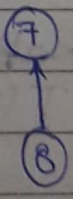
★ (2, 5) → S_5, S_3

$S_1 \cup S_3 = \{1,2,3,4,5,6\} = S_6$.

★ (6, 8) → S_6, S_4

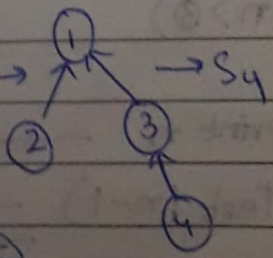
$S_6 \cup S_4 = \{1,2,3,4,5,6,7,8\} = S_7$

both exist in same set, cancelled!

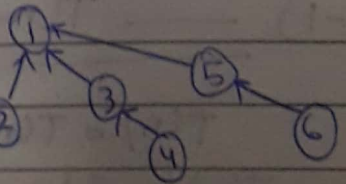


★ (5, 7) → S_7, rejected !

★ (2, 4) $S_1 \cup S_2 \rightarrow$ → S_4



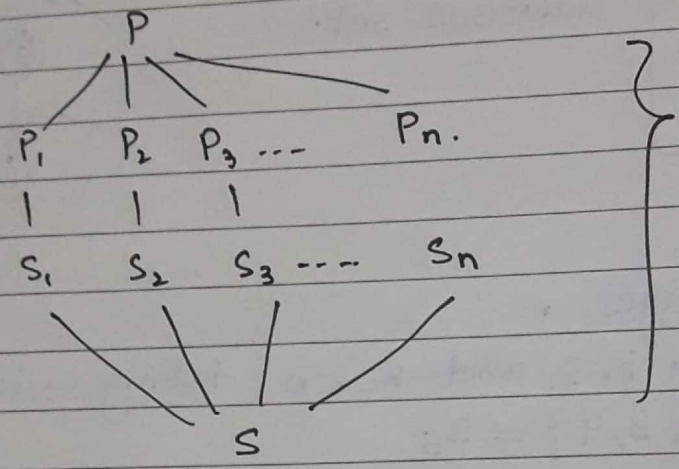
★ (2, 5) $S_4 \cup S_3 \rightarrow$



Dated:

Topic 2 : Divide and Conquer.

→ problem is broken into sub-problems and solutions are combined to get one ~~==~~ solutions.



} Recursive approach.

Algorithms using divide and conquer technique:

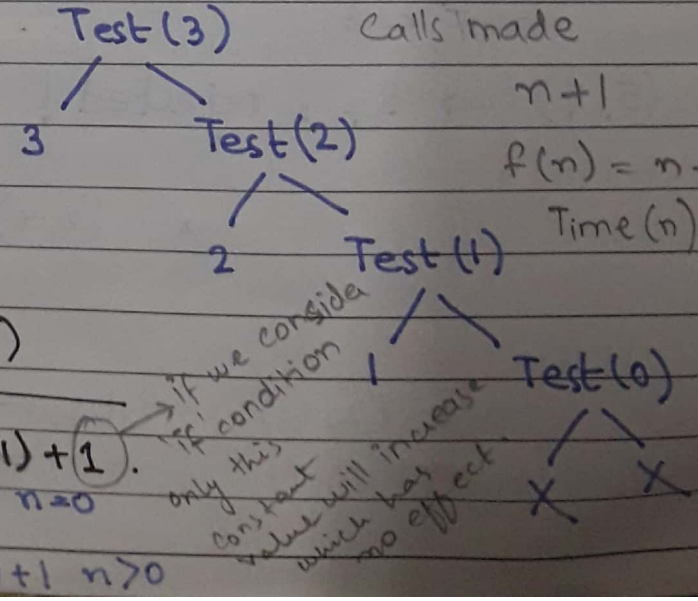
1- Binary Seach.
2- Finding maximum & minimum.
3- Merge Sort.
4- Quick Sort.
5. Strassen's Matrix Multiplication.

2.1.1 Recurrence Relation

```
Test (n) ——— T(n)
{
  if (n>0)
  {
    print n ——— 1
    Test (n-1) ——— T(n-1)
  }
}
```

$T(n) = T(n-1) + 1$.

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



Dated: Finding out time through recurrence relation :

The recurrence relation we get is $T(n) \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$

∴ when $n>0$ $\quad T(n) = T(n-1) + 1$ — (1)

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

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

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

⋮ ⋮ ⋮ $\quad$ putting in eq — (1)

$T(n) = T(n-k) + k$ — continued up to k ~~times/steps~~

when 1 reached $n-k=0$, then, $n=k$.

so,

$$T(n) = T(n-n) + n$$
$$= T(0) + n$$
$$T(n) = n+1$$
$$\Rightarrow O(n) \rightarrow \Theta(n).$$

Dated: 2·1·2

```
Test (n)                    —— T(n)
{
  if (n>0)                  —— 1
  {
    for (i=0; i<n; i++)     —— n+1
    {
      printf (n)            —— n
    }
    Test (n-1)              —— T(n-1)
  }
}
```

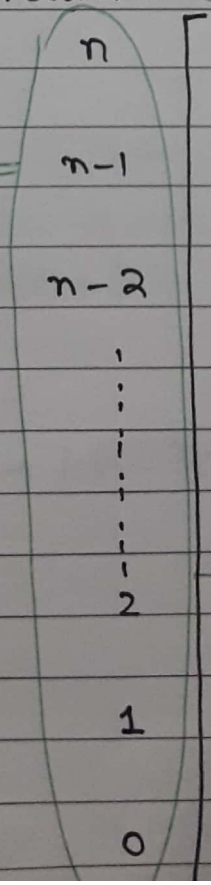
$T(n) = T(n-1) + 2n + 2$ → it's linear so round it to n.

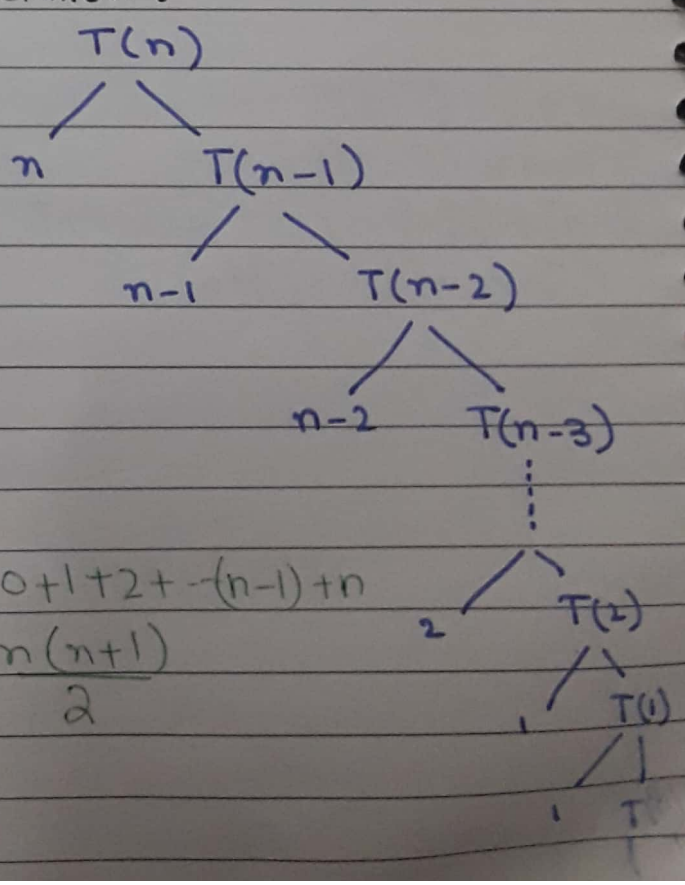
so, $T(n) = T(n-1) + n$.

now,

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

Tree Method of finding recurrance relation:





$\rightarrow 0+1+2+\cdots+(n-1)+n$

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

∴ $T(n) = \frac{n(n+1)}{2}$

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

$= \Theta(n^2)$

Dated:

② Back substitution Method.

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

$T(n) = T(n-1) + n.$

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

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

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

∴ $T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + n-2$.

$T(n-3) = T(n-4) + n-3$.

$= T(n-k) + (n-(k-1)) + (n-(k-2)) + (n-(k-3)) + \cdots + (n-1) + n.$

Assume $n-k=0.$

$n = k.$

So, $T(n) = T(n-n) + (n-n+1) + \cdots (n-1) + n.$

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

$T(n) = \boxed{1 + \frac{n(n+1)}{2}}$ → These are the total no. of calls made.

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

$= \Theta(n^2).$

$n!$ has an upper bound of n^n.

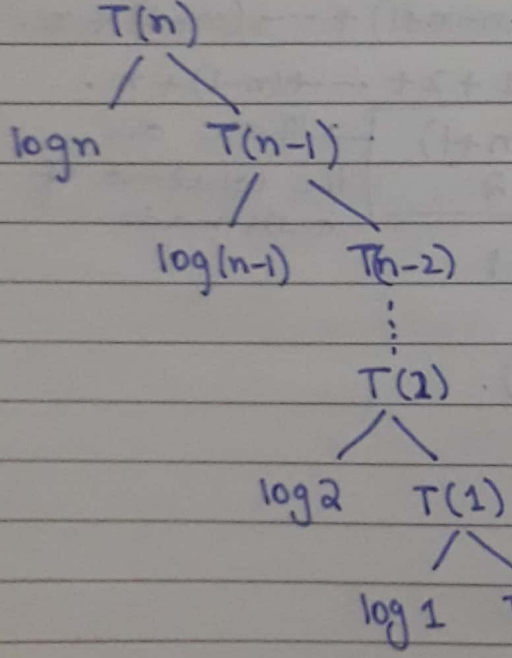
Date: 2.1.3

```
T(n) ——  void Test (n)
         {
1 ——     if(n>0)
         {
log n ——     for(i=1; i<n; i=i*2)
             print i
         }
T(n-1) ——    Test(n-1)
         }
```

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

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

1) Tree method.



— log n

— log(n-1)

— log(2)

— log(1)

∵ $\log a + \log b = \log(ab)$

$\Rightarrow \log n + \log(n-1) + \cdots + \log 2 + \log 1$

so,

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

$= \log(n!)$

$\because 1 \leq n! \leq n^n$

so using upper bound.

$= \log n^n$

$O(n \log n)$

Dated:

② Back Substitution.

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

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

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

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

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

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$

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

Assume $n-k=0$

$n=k.$

$T(n) = T(0) + \log n!$ $\qquad \log n! = \log(1*2\times3\times\cdots(n-1)\times n)$

$= 1 + \log n!$

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

Note :

$\therefore T(n) = T(n-1) + 1 \rightarrow O(n)$

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

$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$

$T(n) = T(n-1) + n^2 \rightarrow O(n^3)$

$T(n) = T(n-2) + 1 \rightarrow n/2 \Rightarrow O(n)$

$T(n) = T(n-100) + n \rightarrow n^2/100 \Rightarrow O(n^2)$

$\therefore$ value of $T(n) = 2T(n-1) + 1$ will be different due to co-efficient.

in G.P : $a + ar + ar^2 + ar^3 + \dots ar^k = \frac{a(r^{k+1}-1)}{r-1}$

for $2^0 + 2^1 + 2^2 \dots 2^k$; $a = 1, r = 2$

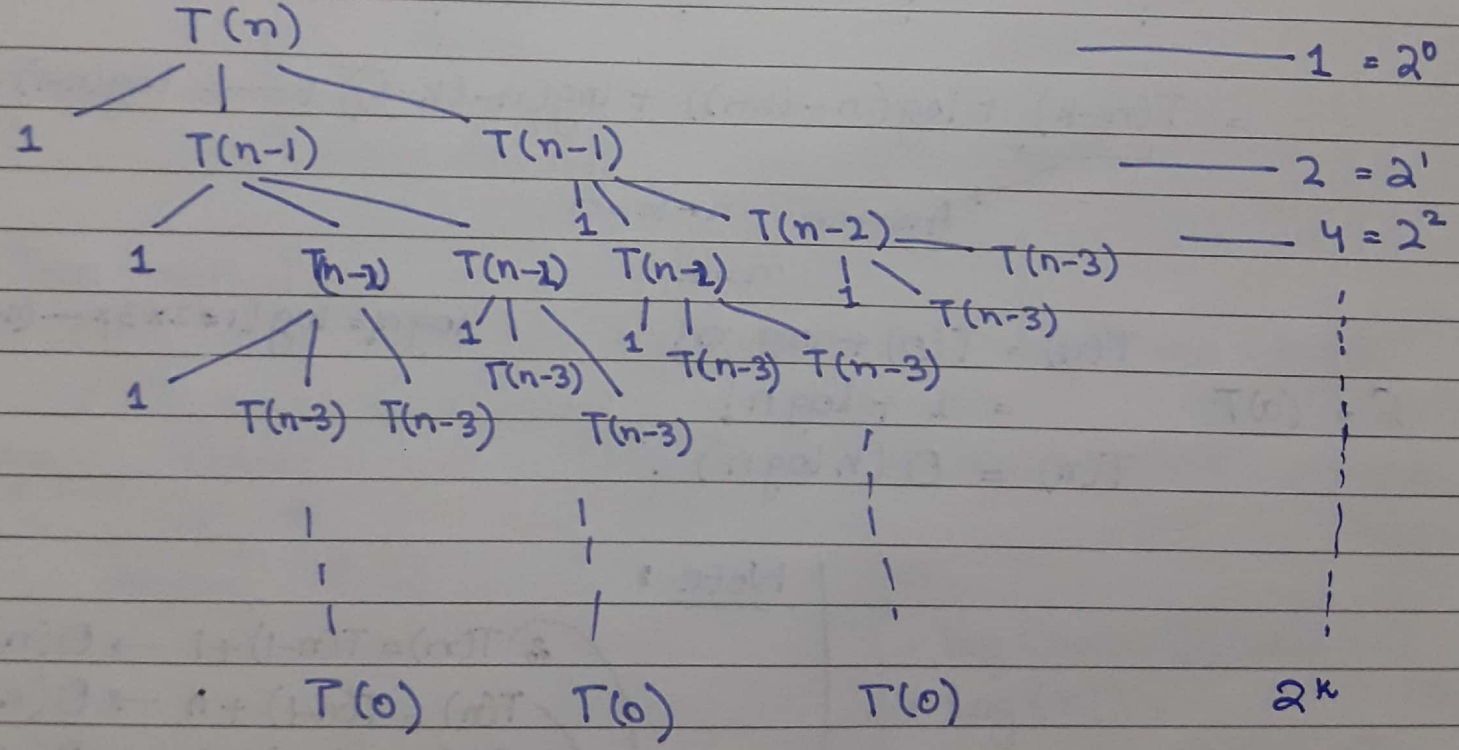
$\Rightarrow \frac{1(2^{k+1}-1)}{2-1} = (2^{k+1}-1)$

Dated: 2.1.4

```
T(n) ——  Test (n)
         {
1 ———    if (n>0)
         {
1 ———    print n
T(n-1)   Test (n-1)
T(n-1)   Test (n-1)
         }
         }
```

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

1) Recursion Tree Method :



$\therefore 1 + 2^1 + 2^2 + 2^3 + \dots 2^k = 2^{k+1} - 1 \quad -①$

Assume $n - k = 0 \Rightarrow n = k$.

so eq-① $\Rightarrow 2^{n+1} - 1$

$= 2^n \cdot 2 - 1$

$= O(2^n)$

Dated:

② Back Substitution Method :

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

$T(n) = 2T(n-1)+1$ —— ①

$= 2[2T(n-2)+1]$ 1.

$T(n) = 2^2 T(n-2) + 2 + 1.$ —— ②

$= 2^3 T(n-3) + 2^2 + 2 + 1.$ —— ③

⋮ ⋮ ⋮

$2^k T(n-k) + [2^{k-1} + 2^{k-2} + \cdots 2^2 + 2^1 + 2^0]$ —— Ⓐ

Assume $n-k=0$.

$n=k$.

$2^k + 2^{k+1} - 1$

$2^k (1+2^1-1)$

Ⓐ $\Rightarrow$ $2^n T(n-n) + 1+2+\cdots\cdots 2^{k-1}$

$= 2^n T(0) + 2^k - 1$

$= 2^n + 2^n - 1 \quad = 2 \cdot 2^n - 1$

$= O(2^n)$

$= \Theta(2^n)$.

Dated:

2.2 Master Theorem For Decreasing Function.

General form of recurrance relation:

$T(n) = aT(n-b) + f(n)$

Assume that

$a>0$, $b>0$ and $f(n) = O(n^k)$ where $k \geq 0$.

when $a=1$, (case 1)

we get: $O(n^{k+1})$ → i.e $O(n)$ has $k=0$.

$O(n * f(n))$

when $a<1$, (case 2)

we get: $O(n^k)$

$O(f(n))$

when $a>1$ (case 3)

we get: $O(n^k . a^{n/b})$

$T(n) = \underset{a}{2}T(n-\underset{b}{3}) + n \Rightarrow O(n.2^{n/3})$

$n^0 . a^{n/b}$

To Remember:

$T(n) = T(n-1) + 1$ —— $O(n)$

$T(n) = T(n-1) + n$ —— $O(n^2)$.

$T(n) = T(n-1) + \log n$ — $O(n \log n)$

$T(n) = 2T(n-1) + 1$ — $O(2^n)$

$T(n) = 3T(n-1) + 1$ — $O(3^n)$.

$T(n) = 2T(n-1) + n$ — $O(n 2^n)$

$T(n) = 2T(n-2) + n$ — $O(n \cdot 2^{n/2})$

examples:

* $T(n) = T(n-1) + 1$

$n \times 1 = n \Rightarrow O(n)$

* $T(n) = T(n-1) + n$

$\Rightarrow O(n^2)$

* $T(n) = T(n-1) + \log n$.

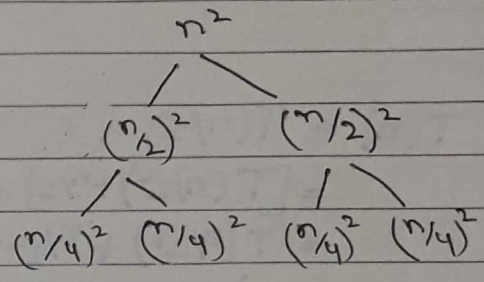
$\Rightarrow O(n \log n)$

Dated: 2.3.1 Recurrence Relation Dividing.

size at level i = $n/2^i$

Cost of each node at level i = $(n/2^i)^2$ —— ①

# of nodes = 2^i —— ②



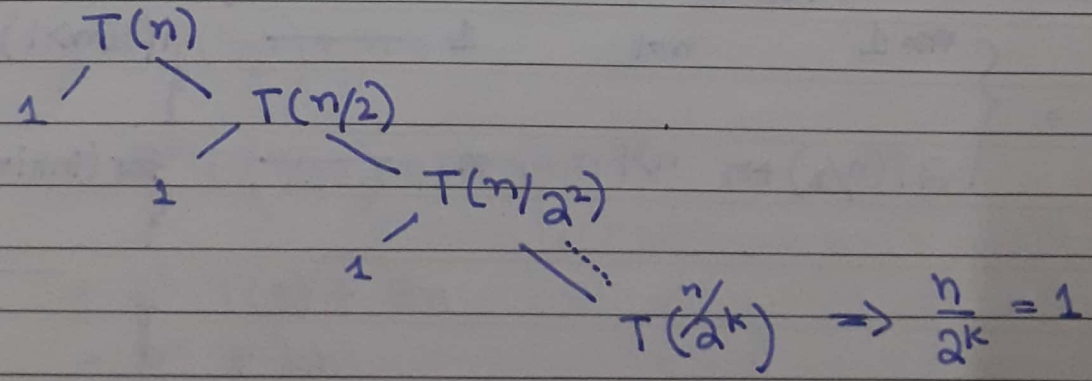
Total cost = ① × ②

$= \left(\frac{n}{2^i}\right)^2 \times 2^i$

$= \frac{n^2}{2^i}$

```
Test(int n)          ——— T(n)
{
  if (n>1)           ——— 1
  {
    print('%d',n)    ——— 1
  }
  Test(n/2)          ——— T(n/2)
}
```

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



$2^k = n$

$k = \log_2 n$

$k = \log n$.

log n

Dated:

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

$T(n) = T(n/2) + n.$

$= [T(n/2^2) + n/2] + n.$

$= T(n/2^3) + n/2^2 + n/2 + n$

$\vdots \quad \vdots \quad \vdots$

$= T(n/2^k) + n/2^{k-1} + \cdots n/2 + 1$

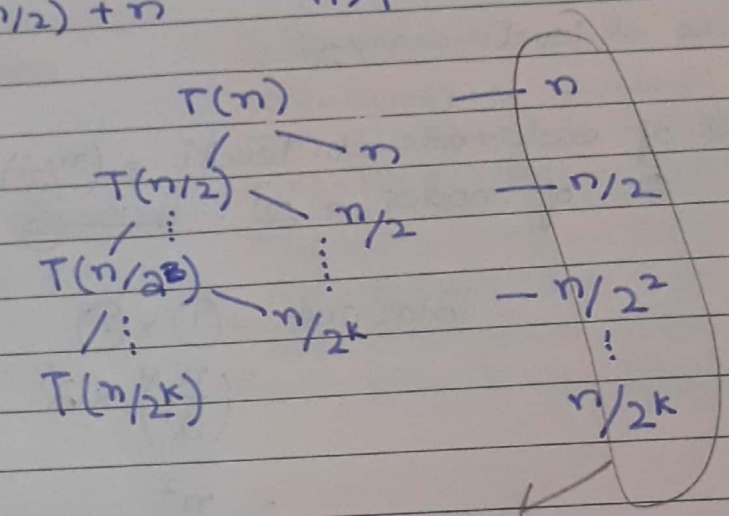
Assume that,

$n/2^k = 1$

$n = 2^k$

$k = \log n$

$T(n) = T(1) + n$



$n + n/2 + n/2^2 + \cdots n/2^k$

$\Rightarrow n(1 + 1/2 + 1/2^2 + \cdots 1/2^k)$

$\Rightarrow n \sum_{i=0}^{k} \frac{1}{2^i}$ → equal to 1.

$\Rightarrow n \times 1 \Rightarrow O(n)$

2.3.3

recurrence relation:

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

$T(n)$ —— Test (int n)

{

1 —— if (n>1)

{

n —— for (int i=0; i<n; i++)

{ —

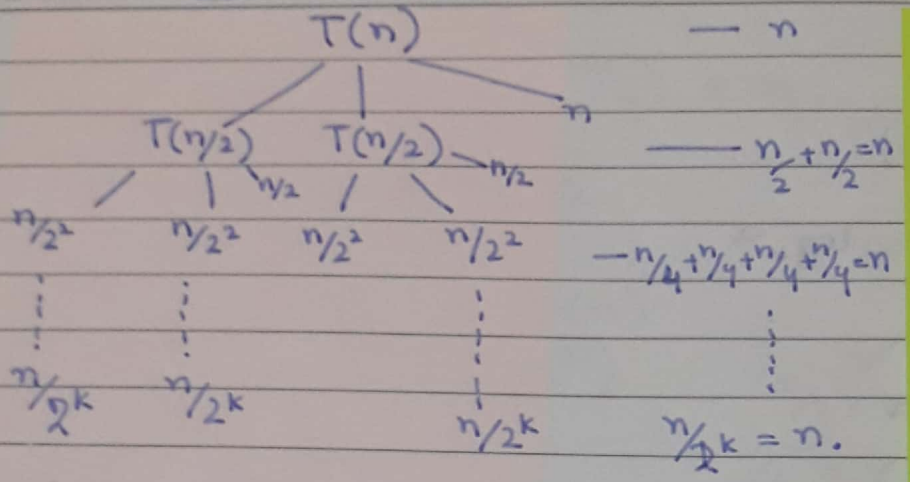
}

$T(n/2)$ $T(n/2)$

$T(n/2)$ $T(n/2)$

$T(n) = 2T(n/2) + n$ }

Dated: Recurrence Tree



∴ k times n

so nk. where for k, $n/2^k = 1$

$k = \log n$.

∴ $n(\log n)$

$\Rightarrow O(n \log n)$.

Substitution method:

$T(n) = 2T(n/2) + n$.

$= 2T(n/2^1) + n$.

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

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

$= 2^3 T(n/2^3) + n + n + n$.

$= 2^k T(1) + 3n$.

$= 2^k + kn$.

$= \log n + n(\log n)$

$O(n \log n)$.

Dated:

Master Theorem:

$$T(n) = aT(n/b) + f(n)$$

Case 1 : if $f(n) > n^{\log_b a}$

$O(f(n))$

Case 2: if $f(n) < n^{\log_b a}$

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

Case 3 : if $f(n) = n^{\log_b a}$

$O(f(n) \cdot \log n)$.

Dated: ^nC_r

0	1	2	3	4	5	6	7	8	9
3	6	8	12	18	31	36	42	55	62

l m h

$m = \left(\frac{l+h}{2}\right) + 1 = \frac{9}{2} + 1 = 4+1 = 5$

n-2 n-1 3

n-1 n 1

① Iterative :

```
BinarySearch(A, n, key)
{
    l = 0,
    h = n-1, mid = ((l+h)/2) + 1
while (l ≤ h)
{
    if (key == A[mid])
        return mid

    if (key < A[mid])
        h = mid - 1

    else
        l = mid + 1
}
```

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

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

$\longrightarrow O(\log n)$

② Recursive

```
BinarySearch(l, h, key)
{
  if(l==h)
  {
    if (A[l] == key)
      return l;
  }
else
{
  mid = ((l+h)/2) + 1

  if (key == A[mid])
  { return mid }
  if (key < A[mid])
  { return BinarySearch(l, mid-1, key
  }
else
  return BinarySearch(mid+1, h, key
}
```

$T(n/2)$

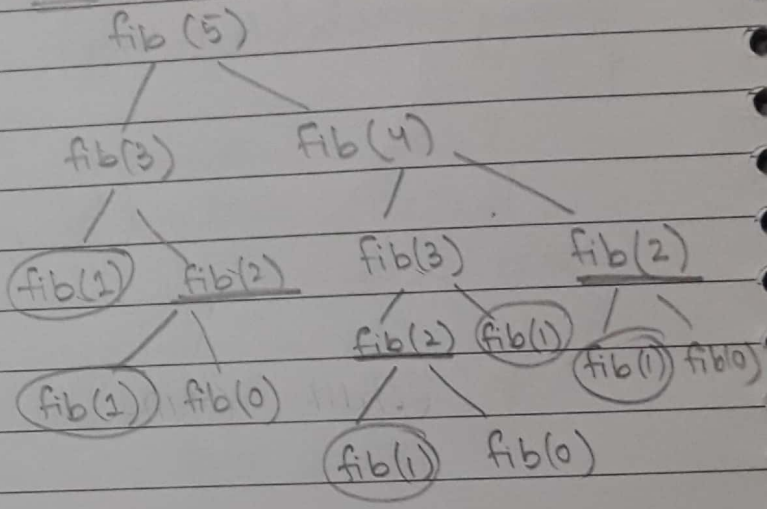
Dated:

Dynamic Programming

1- Greedy Method.
2. Dynamic Programming —— They follow principle of optimality.

```
int fib(int n)
{
   if (n<=1)
     return n
   else
     return fib(n-2) + fib(n-1)
}
```

i.e : n = 5



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

$O(2^n)$.

Memoization follow top down approach.

${}^nC_0$ is 1 and ${}^nC_n = 1$

$\frac{{}^{2n}C_n}{n+1}$ = no. of possible comparisons

```
int BionomialCoeff(int n, int r)
{  if (r==0 || r==n)
   {  return 1
   }
   else
      return BionomialCoeff(n-1, k-1) + (n-1, k)
}
```

$${}^5C_2 = \frac{5\times4}{2} = \frac{20}{2} = 10.$$

$${}^nC_r = \frac{n!}{r!(n-r)!}$$