Ch:1

Asymptotic Notation

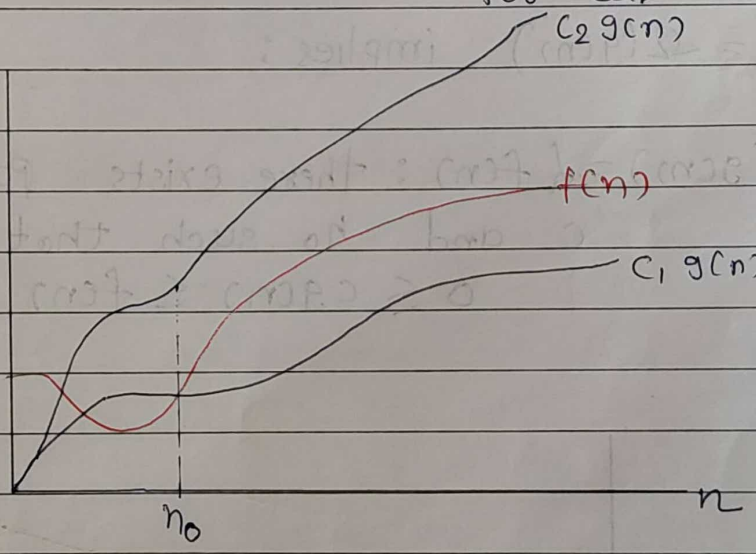
→ When we look at input size large enough to make only the order of growth of the running time relevent, we are studying the asymptotic efficiency of algorithms.

→ **Θ notation :** (tight bound) (average bound)

for a given function $g(n)$,

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$.

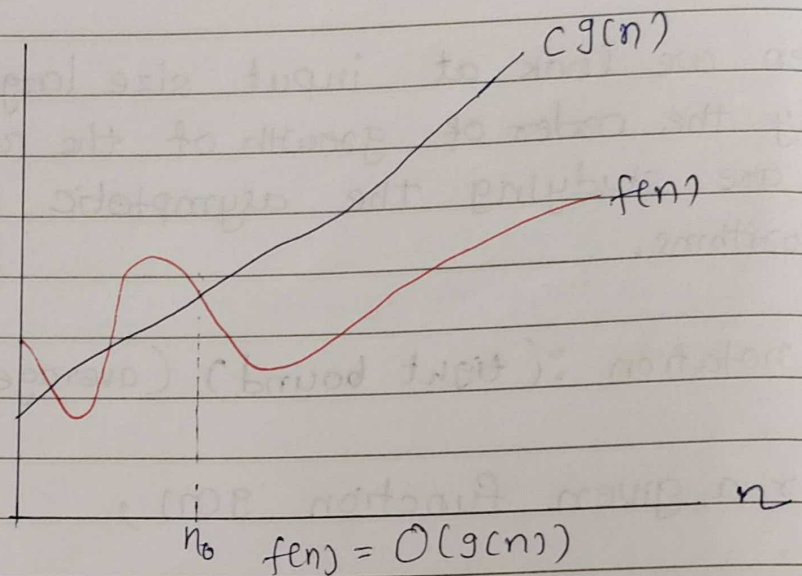


→ **O Notation** (upper bound)

$f(n) = O(g(n))$ implies :

$O g(n) = \{ f(n)$: there exist positive constants c and n_0 such that

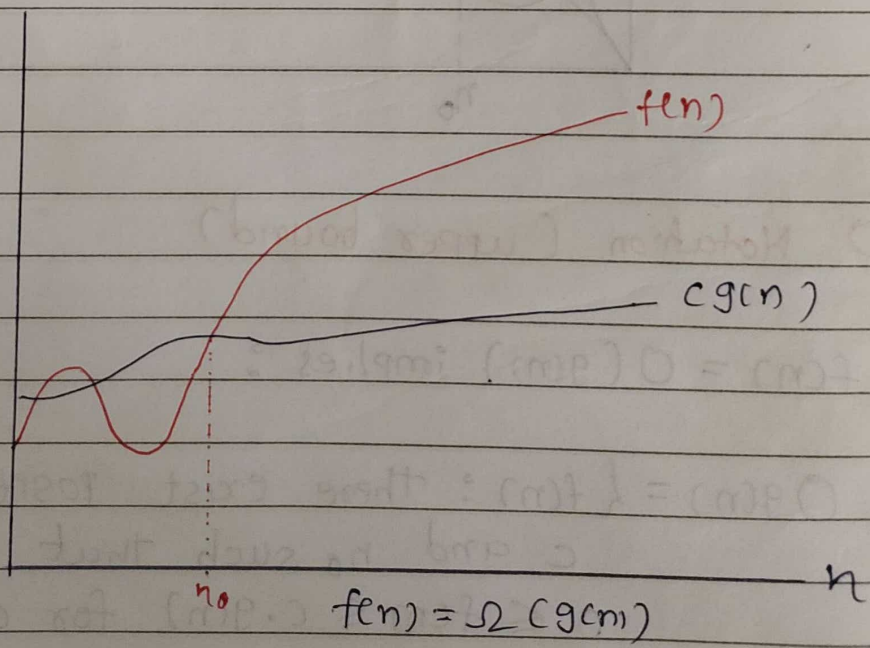
$0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0 \}$



* Ω notation: (Lower bound)

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

$\Omega(g(n)) = \{ f(n)$: there exists positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0 \}$.



Examples :

1) find upper bound for $f(n) = 2n+2$

$$0 \leq f(n) \leq c \cdot g(n)$$

$$2n+2 \leq c \cdot g(n)$$

$2n+2 \leq 4n$ for $n \geq 1$

↳ instead of 4 you can ~~sing~~ write any number which is greater than 4

So: $f(n) = O(n)$ ✓ $f(n) = O(2^n)$ ✓ [try to write closest function]

$f(n) = O(n^2)$ ✓

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

Lower bound — average bound — upper bound.

2) $f(n) = 2n^2+3n+4$ find upper bound

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

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

$2n^2+3n+4 \leq 9n^2$ for all $n \geq 1$

$n_0 = 1$

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

Lower bound

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

$c \cdot g(n) \leq 2n^2+3n^2+4n^2$

$1 \cdot n^2 \leq 2n^2+3n^2+4n^2$ for all $n \geq 1$

$n_0 = 1$

$\therefore f(n) = \Omega(n^2)$

you can't place n! to any fix point.
(it may place near to n or n^n)

tight found.

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

$1 \cdot n^2 \leq 2n^2+3n+4 \leq 9n^2 \quad n \geq 1$

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

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

~~upper bound~~.

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

$\therefore f(n) = O(n^2 \log n)$
$f(n) = \Theta(n^2 \log n)$
$f(n) = \Omega(n^2 \log(n))$

* $f(n) = n!$ ⇒

$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

$c_1 g(n) \leq n! \leq c_2 \cdot g(n)$

$0.1 \times 1 \times 1 \times 1 \leq 1 \times 2 \times 3 \times \ldots \times n \leq 0 n \times n \times n \times n \times n \ldots n$

$1 \leq n! < n^n$

$f(n) = \Omega(1)$
$f(n) = O(n^n)$

we can't find ~~bound~~ Θ for n! because we don't have same g(n).

* f(n) = log n!

log(1×1×1×1) log(1×2×3×...×n) ≤ log(n×n×n×...n)

0 but we right

1

1 ≤ log n! ≤ log n^n ∴ no tight bound.

So for factorial we can't find tight bound.

=> Properties of Asymptotic Notation

→ General properties:

→ if f(n) is O(g(n)) then a * f(n) is O(g(n)).

e.g: $f(n) = 2n^2 + 5$ is $O(n^2)$

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

$= 14n^2 + 35$

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

→ if f(n) is Ω(g(n)) then a * f(n) is Ω(g(n))

→ if f(n) is Θ(g(n)) then a * f(n) is Θ(g(n))

→ Refexive: ~~applicable for~~ for O and Ω also

if f(n) is given the f(n) is O(f(n))

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

$O(n^2)$

* Transitive (for ~~all~~ also for Θ, Ω)

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

e.g: $f(n) = n$ $g(n) = n^2$ $h(n) = n^3$

n is $O(n^2)$ and n^2 is $O(n^2)$ the n is $O(n^3)$

* Symmetric Properties: (true only for Θ)

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

e.g. $f(n) = n^2$ and $g(n) = n^2$
then $f(n) = \Theta(n^2)$ and $g(n) = \Theta(n^2)$

* Transpose symmetric (true for O and Ω)

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

e.g $f(n) = n$ $g(n) = n^2$
then ~~n~~ is $f(n) = O(n^2)$
so $g(n) = \Omega(n) = \Omega(f(n))$

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

* 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 $f(n) = n$, $d(n) = n^2$
$f(n) = O(n)$ $d(n) = O(n^2)$
$f(n) + d(n) = n + n^2 = O(n^2)$

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

Time Complexity

* ~~Recurrence Relation~~ :

* Frequency count Method.

* Algorithm Sum(A, n)

```
{
    s=0;
    for (i=0; i<n; i++)
    {
        s = s+A[i];
    }
    return s;
}
```

Here, we are assuming that, ~~1 step~~ for executing 1 step it will require 1 unit of time.

$s=0$ ——————— 1

for $\underbrace{(i=0;}_{1}$ $\underbrace{i<n;}_{n+1}$ $\underbrace{i++)}_{n}$ ⟶ $n+1$

$s=s+A[i]$ ⟶ n

return s ⟶ 1

———————

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

$f(n) = O(n)$

classmate
Date
Page

space complexity

A → n
n → 1
s → 1
i → 1
$S(n) = n+3$
$S(n) = O(n)$

* Sum of two matrices

Algo:

```
Add (A,B,n)                    n×n size
{
  for (i=0; i<n; i++)  ⟶ n+1
  {
    for (j=0; j<n; j++) → (n+1)*n   } n
    {
      C[i,j] = A[i,j] + B[i,j] ; n*n
    }
  }
```

time function $f(n) = (n+1) + (n+1)*n + n^2$
$= n+1+n^2+n+n^2$
$= 2n^2+2n+1$
$t(n) = O(n^2)$

space:
A → n×n
B → n×n
C → n×n
n → 1
i → 1
j → 1
$2n^2+3 = O(n^2)$

Matrix multiplication

```
Algo multiply (A, B, n)
{
    for (i=0; i<n; i++)                    n+1
    {
        for (j=0; j<n; j++)                n*(n+1)
        {
            c[i,j] = 0;                    n*n
            for (k=0; k<n; k++)            n*n*(n+1)
            {
                c[i,j] = c[i,j] + A[i,k] * B[k,j];   n*n*n
            }
        }
    }
}
```

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

$f(n) = O(n^3)$ (degree of Polynomial of $f(n) = n^3$)

Space:

A → n^2
B → n^2
C → n^2
n → 1
i → 1
j → 1
K → 1

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

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

* for (i=0 ; i<n ; i++) → n+1
  {
  stmt ; → n
  }
  2n+1

→ O(n)

* for (i=n ; i>0 ; i--) → n+1
  {
  stmt ; → n.
  }
  O(n)

* for (i=1 ; i<n ; i=i+2) → n/2 +1
  {
  stmt ; → n/2
  }
  f(n) = n + n/2

  f(n) = O(n)

* for (i=1 ; i<n ; i=i+20)
  {
  stmt ; → n/20
  }

although f(n) = O(n)

```
* for (i=0; i<n; i++)          → n+1
  {
      for (j=0; j<n; j++)      → n*(n+1)
      {
          stmt;                → n*n
      }
  }
```

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

```
* for (i=0; i<n; i++)
  {
      for (j=0; j<i; j++)
      {
          stmt;
      }
  }
```

i	j	no. of times execute stat.
0	0	0 times
1	0 ✓	1 time
	1 ✗	
2	0 ✓	2 times
	1 ✓	
	2 ✗	
3	0	3 times
:	1	
:	2	
	3 ✗	
n		n times

total = $0+1+2+3+\cdots+n$

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

$= O(n^2)$

```
p=0;
for (i=1; p<=n; i++)
{
    p=p+i;
}
```

condition is depend on i

i	p
1	0+1=1
2	1+2=3
3	1+2+3=
4	1+2+3+4
⋮	⋮
k	$1+2+3+4+\cdots+k = \frac{k(k+1)}{2}$

why here k times?

→ we don't know how many time it loop is executing

Assume $p > n$

$\therefore p = \frac{k(k+1)}{2}$ $\quad \frac{k(k+1)}{2} > n$

$\approx k^2 > n$

$k > \sqrt{n}$

$f(n) = O(\sqrt{n})$

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

i
1
1*2
$2*2 = 2^2$
$2^2*2 = 2^3$
⋮
2^k

or

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

$2^k = n$

$k = \log_2 n$

Assume $i \geq n$

$2^k \geq n$

$2^k = n$

$k = \log_2 n$

$O(\log_2 n)$

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

$i = 1 + 1 + 1 + 1 + \cdots + 1 = n$ (k times)

$k = n$

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

i
n
$\frac{n}{2}$
$\frac{n}{2^2}$
$\frac{n}{2^3}$
$\frac{n}{2^4}$
⋮
$\frac{n}{2^k}$

assume $i < 1$

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

$n < 2^k$ $\quad k = \log_2 n$

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

i*i<n , it will terminate i*i >=n

$i^2 >= n$

$i >= \sqrt{n}$

```
for (i=0; i<n; i++)
  { stmt;          ⟶ n
  }
for (j=0; j<n ; j++)
  {
    stmt1;         ⟶ n
  }
```

f(n) = 2n

f(n) = O(n)

```
P=0
for (i= 1; i<n ; i = i*2)
  {
     P++;          ⟶ log n
  }
  for (j=0 ; j<P ; j=j*2)
     {stmt ;       ⟶ log(P) → log(log n)
     }
```

$p = \log(n)$

```
for(i=0; i<n; i++)          → (n+1) ≈ n
{
    for(j=1; j<n; j=j*2)    → n * (log n)
    {
        stmt;               → n * (log n)
    }
}
```

$2n\log n + n$

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

Summery

for (i=0

for (i=0; i<n; i++) → $O(n)$

for (i=0; i<n; i=i+2) → $n/2 \rightarrow O(n)$

for (i=n; i>1; i--) → $O(n)$

for (i=1; i<n; i=i*2) → $O(\log_2 n)$

for (i=1; i<n; i=i*3) → $O(\log_3 n)$

for (i=n; i>1; i=i/2) → $O(\log_2 n)$

Analysis of if & while.

* for i=1 to n do
{
    stmt;
}

* i=0 ——— 1
while (i<n) ——→ n+1
{
    stmt; ——→ n
    i++; ——→ n
}

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

* a=1
while(a<b)
{
    stmt;
    a=a*2;
}

there is no n is so we don't know how many time loop will execute.
→ It will depend on condition.

a	terminate
1	$a \geq b$
$1*2 = 2$	$a = 2^k$
$2*2 = 2^2$	$2^k \geq b$
$2^2*2 = 2^3$	$k = \log_2 b$
⋮	

2^k (assuming that it stop at 2^k)

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

→ log(n)

```
i = 1
k = 1
while (k < n)
{
    stmt;
    k = k + i;
    i++;
}
```

or

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

i	k
1	1
2	1+1 = 2
3	2+2
4	2+2+3
5	2+2+3+4
⋮	⋮
m	2+2+3+4+⋯ m

$1+1+2+3+\cdots+m$

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

$\simeq \frac{m(m+1)}{2}$

It terminate

$k \geq n$

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

$\simeq m^2 \geq n$

$m = \sqrt{n}$

$f(n) = O(\sqrt{n})$

Recurrence Relation

How

* Defination

A Recurrence relation is an equation that recursively defines a sequence where the next term is a function of the ~~Prever~~ Previous term.

* How to Write Recurrence Relation

→ ~~Dividing~~ Decreasing Function

(1)

```
void Test(int n)
{
    if(n>0)
    {
        Printf("%d", n)
        Test(n-1);
    }
}
```

→ step 1: Assume that Time taken by Test(n) is T(n)

→ T(n) will be total time taken by the algo.

if(n>0) → Ignoring The condition

Printf("%d", n) → takes 1 unit of time

Test(n-1) will take T(n-1)

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

let's write Recurrence Relation

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

here if n=0 then we are not doing anything so I have taken 1 (Constant)

* Void Test(int n) ⟶ T(n)

```
{
  if(n>0)                          ——  1
  {
    for(i=0; i<n; i++)             ⟶  n+1
    {
      Printf("%d", n);             ⟶  n
    }
    Test(n-1);                     ⟶  T(n-1)
  }
}
```

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

↓

apply asymptotic notation

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

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

```
Void Test(int n)                    ⟶ T(n)
{
   if(n>0)                          ⟶ 1
   {
      for(i=1; i<n; i=i*2)          → logn + 1
      {
         Printf("%d",i);            → logn
      }
      Test(n-1);                    → T(n-1)
   }
}
```

~~T(n)~~ =

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

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

Recurrence Relation

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

```
Algo Test (int n)                   → T(n)
   {
      if(n>0)                       → 1
      {
         Printf("%d", n);           → 1
         Test(n-1);                 → T(n-1)
      } Test(n-1);                  → T(n-1)
   }
```

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

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

Recurrence Relation

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

Dividing Function

```
Algo Test(int n)                    → T(n)
{
    if (n>1)                        → 1
    {
        printf("%d", n);            → 1
        Test(n/2);                  → T(n/2)
    }
}
```

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

Recurrence Relation

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

Solving Recurrence Relation

1) Substitution (Backward substitution)

2) Master's method

3) Recursion Tree method.

Substitution (Backward Substitution)

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

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

find value of $T(n-1)$ using equation (1)

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

put value of $T(n-1)$ in equation (1)

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

assume that continue for k times

$$T(n) = T(n-k) + k \quad \text{---} (3)$$

Recurrence relation

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

$T(n) = T(n-k) + k$

at one point we reach at $n=0$

∴ assume that $n-k=0$

$n=k$

∴ $T(n) = T(n-n) + n$

$T(n) = T(0) + n$

$T(n) = n+1$

$\boxed{T(n) = O(n)}$

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

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

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

∴ $T(n) = T(n-2) + (n-1) + n$

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

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

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

$\vdots$

continue for k times

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

at one point we reach at $n = 0$.

$$\therefore n - k = 0$$
$$n = k.$$

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

$$T(n) = T(0) + 1 + 2 + 3 + \cdots + (n-1) + n$$

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

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

$$\boxed{T(n) = O(n^2)}$$

$\log ab = \log a + \log b$

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

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

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

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

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

⋮

Continue for k times

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

$\because n - k = 0$

$n = k$

$T(n) = T(0) + \log 1 + \log 2 + \log 3 + \ldots + \log(n-1) + \log n$

$= 1 + \log(1 * 2 * 3 * \ldots * n)$

$= 1 + \log n!$

$T(n) = O(\log n!)$ → upperbound is n^n

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

$= O(n \log n)$

Summery for Decreasing Function

$T(n) = T(n-1) + 1 \longrightarrow O(n)$ → multiply by n

$T(n) = T(n-1) + n \longrightarrow O(n^2)$ → multiply by n

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

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

$T(n) = T(n-2) + 1 \longrightarrow \frac{n}{2} = O(n)$

$T(n) = T(n-100) + n = O(n^2)$

GP series:

$1 + 2 + 2^2 + 2^3 + \ldots + 2^k = 2^{k+1} - 1$

$a + ar + ar^2 + \ldots + ar^k = a\left(\frac{r^{k+1} - 1}{r-1}\right)$

$a = 1, r = 2$

$= 2 \cdot 1 \left(\frac{2^{k+1} - 1}{2-1}\right)$

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

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

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

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

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

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

$T(n) = 2^2 [2T(n-3) + 1] + 2 + 1$

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

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

Assume $n - k = 0$

$n = k$

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

$= 2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2 + 1$

$= 1 + 2 + 2^2 + \ldots + 2^{n-2} + 2^{n-1} + 2^n$ → GP series

$= 2^{n+1} - 1$

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

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

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

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

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

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

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

Repeat till k times

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

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

$$n = 2^k$$

$$\log_2 n = k \log_2 2$$

$$k = \log n$$

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

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

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

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

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

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

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

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

~~$T(1)$~~ +

$$= T\left(\frac{n}{2^k}\right) + n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{k-2}} + \frac{1}{2^{k-1}}\right.$$

Page No.
Date

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

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

$T(n) = T(n-4) + (n-2)^2 + n^2$

$T(n) = T(n-6) + (n-4)^2 + (n-2)^2 + n^2$

Repeat k times

$T(n) = T(n-k) + (n-(k-2))^2 + (n-(k-4))^2 + \cdots$

$(n-2)^2 + n^2$

$n - k = 0$

$k = n$

$T(n) = T(0) + 2^2 + 4^2 + 6^2 + 8^2 + \cdots + (n-2)^2 + n^2$

$= 1 + 2^2 [1 + 2^2 + 3^2 + 4^2 + \cdots n^2]$

$= 1 + 2^2 \left(\frac{n(n+1)(2n+1)}{6} \right)$

$= O(n^3)$

Page No.
Date

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

$T(n) = T(n-2) + \log n$ —①

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

$T(n) = T(n-6) + \log(n-4) + \log(n-2) + \log n$

Repeat k times

$T(n) = T(n-k) + \log(n-(k-2)) + \log(n-(k-4)) + \log(n-(k-6)) + \cdots + \log(n-2) + \log n$

$n - k = 0$

$k = n$

$T(n) = T(0) + \log 2 + \log 4 + \log 6 + \cdots + \log(n-2) +$

$= 1 + \log(2 * 4 * 6 * 8 \times 10 \cdots n)$

$= 1 + \log(2(1 * 2 \times 3 \times 4 \times \cdots n))$

$= 1 + \log 2 + \log(1 * 2 \times 3 \times 4 \times \cdots \times n)$

$= 1 + \log 2 + \log(n!)$

$= 1 + \log + n \log n$

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

Formulas:

Logarithms:

$$\log x^y = y \log x$$

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

$$\log^k n = (\log n)^k.$$

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

$$\log \frac{x}{y} = \log x - \log y.$$

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

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Arithmetic Series

$$\sum_{K=1}^{n} K = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Geometric Series

$$\sum_{K=1}^{n} x^K = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1} \quad (x \neq 1)$$

OR $\frac{1 - x^{n+1}}{1 - x}$

Harmonic Series

$$\sum_{K=1}^{n} \frac{1}{K} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \log n.$$

other Formula

$$\sum_{K=1}^{n} \log n = \log 1 + \log 2 + \cdots + \log n = n \log n$$

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

$k = \log_2 n$

$\therefore T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \left[1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^{\log n - 1}}\right]$

$2^{\log_2 n} = n \quad \left(a^{\log_b x} = x^{\log_b a} = n^{\log_2 2} = n\right)$

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

$= T(n/n) + \left[1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^{\log n - 1}}\right]$

Geometric series

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

$= 1 + \frac{\frac{1}{2^{\log n}} - 1}{\left(-1/2\right)}$

$= 1 + \frac{\frac{1}{n} - 1}{-1/2} = 1 - 2\left(\frac{1 - n}{2n}\right)$

$= 1 - 2 +$

$$T(n) = 1 + n\left[\frac{1 - \frac{1}{n}}{1/2}\right]$$

(took minus from)

$$= 1 + n\left[2\left[\frac{n-1}{n}\right]\right]$$

$$= 1 + n\left[\frac{2n}{n} - \frac{2}{n}\right]$$

$$= 1 + n\left[2 - \frac{2}{n}\right]$$

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

$$= 2n - 1$$

$$T(n) = O(n)$$

Page No.
Date

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

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

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

$$T(n/4) = 2T(n/8) + \left(\frac{n}{4}\right)^2$$

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

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

$$= 2^2\left[2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2\right] + \frac{n^2}{2} + n^2$$

$\frac{4}{16}$

$$= 2^3 T\left(\frac{n}{8}\right) + \frac{n^2}{2^2} + \frac{n^2}{2} + n^2$$

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

$$= 2^3\left[2T\left(\frac{n}{16}\right) + \frac{n^2}{64}\right] + \frac{n^2}{2^2} + \frac{n^2}{2} + n^2$$

$$= 16\,T\left(\frac{n}{16}\right) + \frac{n^2}{2^3} + \frac{n^2}{2^2} + \frac{n^2}{2} + n^2$$

$$= 2^4\,T\left(\frac{n}{2^4}\right) + \frac{n^2}{2^3} + \frac{n^2}{2^2} + \frac{n^2}{2} + n^2$$

$$2^k = 1$$

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

$$= 2^k\,T\left(\frac{n}{2^k}\right) + n^2\left[1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^{k-1}}\right]$$

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

$$n = 2^k$$

$$\log n = k \log 2$$

$$\boxed{k = \log_2 n}$$

$$= n\,T(1) + n^2\left[1 + \frac{1}{2} + \cdots \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^{\log_2 n - 1}}\right]$$

Geometric series

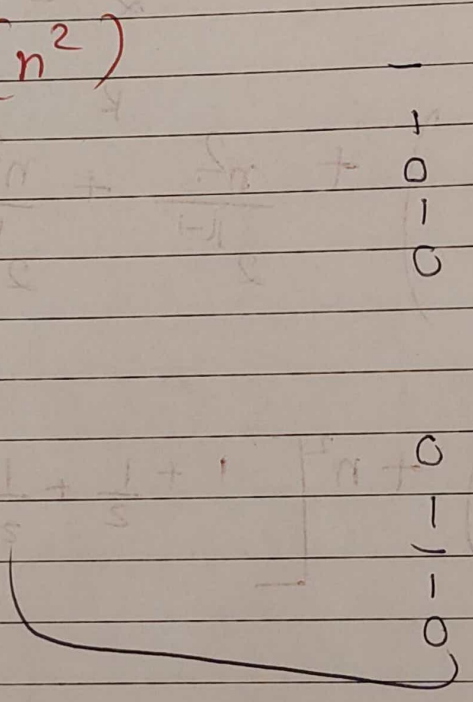
$$T(n) = n + n^2 \left[\frac{1 - 1/n}{1/2} \right]$$

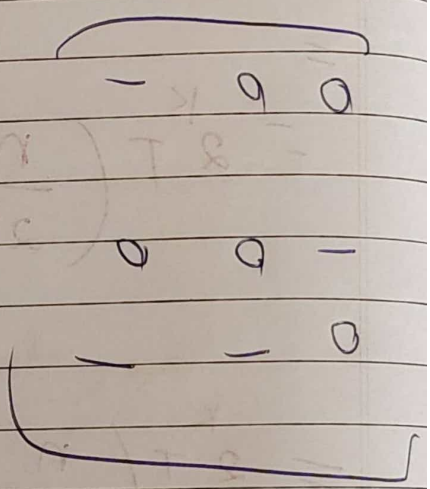
$$= n + n^2 \left[\frac{2 - 2}{n} \right]$$

$$= n + 2n^2 - 2n$$

$$= 2n^2 - n$$

$$= O(n^2)$$





* Master Theorem for Divide and Conquer

If the recurrence is of the form

$$T(n) = aT(n/b) + \Theta(n^k \log^p n),$$

where $a \geq 1$, $b > 1$, $k \geq 0$ and p is Real number,

then :

1) if $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$

2) if $a = b^k$

a. If $p > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

b. If $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log\log n)$

c. If $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

3) If $a < b^k$

a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$

b. If $p < 0$, then $T(n) = O(n^k)$

Examples:

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

$T(n) = aT(n/2) + n^k \log^p n$

here $a=3,\ b=2\ \ k=2\ \ p=0$

$b^k = 2^2 = 4$

$\therefore a < b^k$ case 3 will be applicable

here $p=0$

If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$

$= \Theta(n^2 \log^0 n)$

$= \Theta(n^2)$

If the recurrence is of the form $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, where $a \geq 1, b > 1, k \geq 0$ and p is a real number, then:

1) If $a > b^k$, then $T(n) = \Theta(n^{log_b^a})$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta(n^{log_b^a} log^{p+1} n)$
   b. If $p = -1$, then $T(n) = \Theta(n^{log_b^a} loglogn)$
   c. If $p < -1$, then $T(n) = \Theta(n^{log_b^a})$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k log^p n)$
   b. If $p < 0$, then $T(n) = \Theta(n^k)$

1.22 Problems on Divide and Conquer Master Theorem

For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

Problem-1 $T(n) = 3T(n/2) + n^2$

Solution: $T(n) = 3T(n/2) + n^2 \Rightarrow T(n) = \Theta(n^2)$ (Master Theorem Case 3.a)

Problem-2 $T(n) = 4T(n/2) + n^2$

Solution: $T(n) = 4T(n/2) + n^2 \Rightarrow T(n) = \Theta(n^2 logn)$ (Master Theorem Case 2.a)

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

Solution: $T(n) = T(n/2) + n^2 \Rightarrow \Theta(n^2)$ (Master Theorem Case 3.a)

Problem-4 $T(n) = 2^n T(n/2) + n^n$

Solution: $T(n) = 2^n T(n/2) + n^n \Rightarrow$ Does not apply (a is not constant)

Problem-5 $T(n) = 16T(n/4) + n$

Solution: $T(n) = 16T(n/4) + n \Rightarrow T(n) = \Theta(n^2)$ (Master Theorem Case 1)

Problem-6 $T(n) = 2T(n/2) + nlogn$

Solution: $T(n) = 2T(n/2) + nlogn \Rightarrow T(n) = \Theta(nlog^2 n)$ (Master Theorem Case 2.a)

Problem-7 $T(n) = 2T(n/2) + n/logn$

Solution: $T(n) = 2T(n/2) + n/logn \Rightarrow T(n) = \Theta(nloglogn)$ (Master Theorem Case 2.b)

Problem-8 $T(n) = 2T(n/4) + n^{0.51}$

$a=1, b=2 \quad k=1, p=0$

$a < b^k$.

$n^k log^p n$

$O(n)$

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

$a < b^k$

n^2

Solution: $T(n) = 2T(n/4) + n^{0.51} \Rightarrow T(n) = \Theta(n^{0.51})$ (Master Theorem Case 3.b)

Problem-9 $T(n) = 0.5T(n/2) + 1/n$

Solution: $T(n) = 0.5T(n/2) + 1/n \Rightarrow$ Does not apply ($a < 1$)

Problem-10 $T(n) = 6T(n/3) + n^2 \log n$

Solution: $T(n) = 6T(n/3) + n^2 \log n \Rightarrow T(n) = \Theta(n^2 \log n)$ (Master Theorem Case 3.a)

Problem-11 $T(n) = 64T(n/8) - n^2 \log n$

Solution: $T(n) = 64T(n/8) - n^2 \log n \Rightarrow$ Does not apply (function is not positive)

Problem-12 $T(n) = 7T(n/3) + n^2$

Solution: $T(n) = 7T(n/3) + n^2 \Rightarrow T(n) = \Theta(n^2)$ (Master Theorem Case 3.as)

Problem-13 $T(n) = 4T(n/2) + \log n$

Solution: $T(n) = 4T(n/2) + \log n \Rightarrow T(n) = \Theta(n^2)$ (Master Theorem Case 1)

Problem-14 $T(n) = 16T(n/4) + n!$

Solution: $T(n) = 16T(n/4) + n! \Rightarrow T(n) = \Theta(n!)$ (Master Theorem Case 3.a)

Problem-15 $T(n) = \sqrt{2}T(n/2) + \log n$

Solution: $T(n) = \sqrt{2}T(n/2) + \log n \Rightarrow T(n) = \Theta(\sqrt{n})$ (Master Theorem Case 1)

Problem-16 $T(n) = 3T(n/2) + n$

Solution: $T(n) = 3T(n/2) + n \Rightarrow T(n) = \Theta(n^{\log 3})$ (Master Theorem Case 1)

Problem-17 $T(n) = 3T(n/3) + \sqrt{n}$

Solution: $T(n) = 3T(n/3) + \sqrt{n} \Rightarrow T(n) = \Theta(n)$ (Master Theorem Case 1)

Problem-18 $T(n) = 4T(n/2) + cn$

Solution: $T(n) = 4T(n/2) + cn \Rightarrow T(n) = \Theta(n^2)$ (Master Theorem Case 1)

Problem-19 $T(n) = 3T(n/4) + n \log n$

Solution: $T(n) = 3T(n/4) + n \log n \Rightarrow T(n) = \Theta(n \log n)$ (Master Theorem Case 3.a)

Problem-20 $T(n) = 3T(n/3) + n/2$

Solution: $T(n) = 3T(n/3) + n/2 \Rightarrow T(n) = \Theta(n \log n)$ (Master Theorem Case 2.a)

* Recursion Tree method

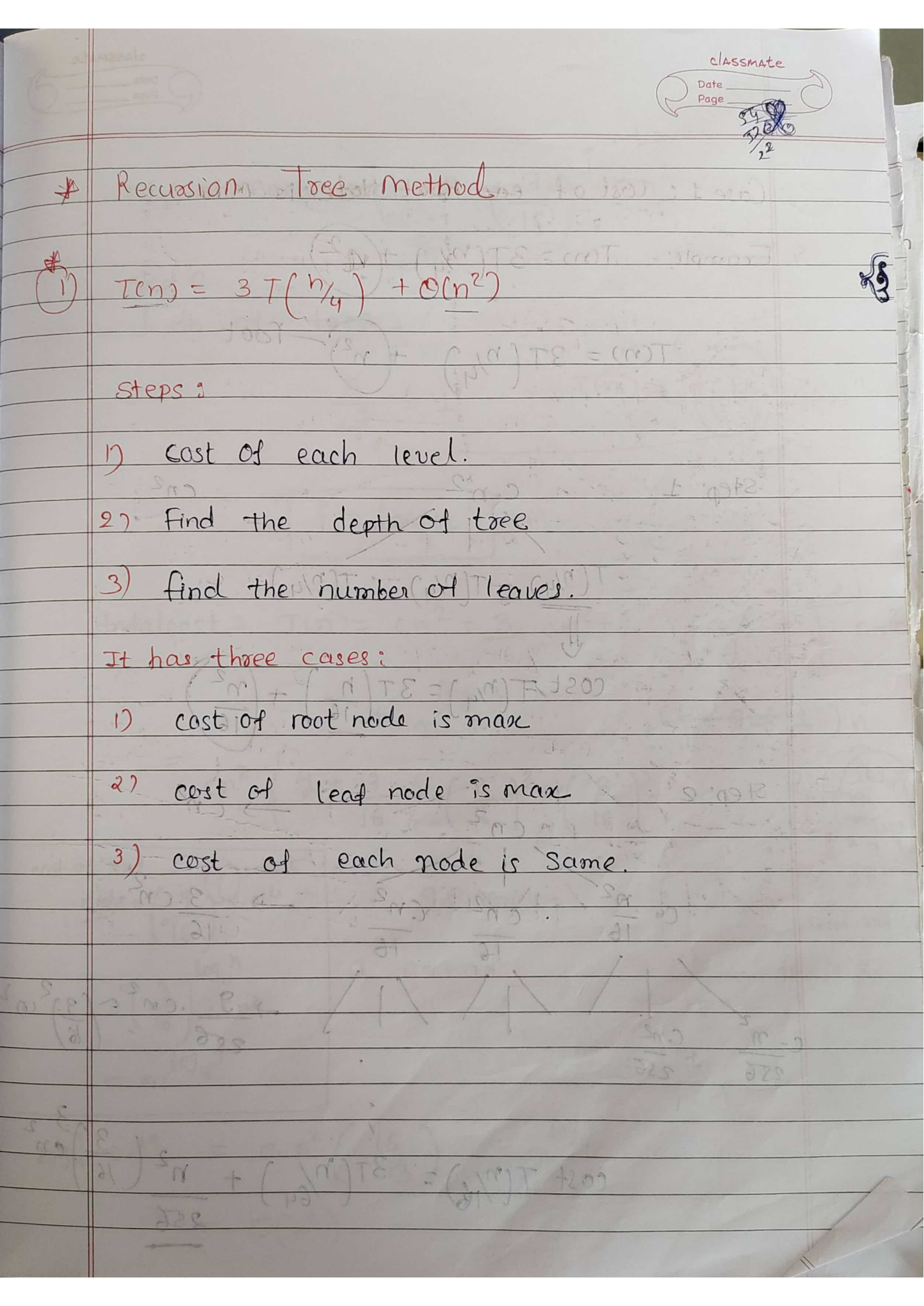
(1) $T(n) = 3T(n/4) + O(n^2)$

Steps :

1) Cost of each level.
2) Find the depth of tree
3) find the number of leaves.

It has three cases :

1) cost of root node is max
2) cost of leaf node is max
3) cost of each node is same.

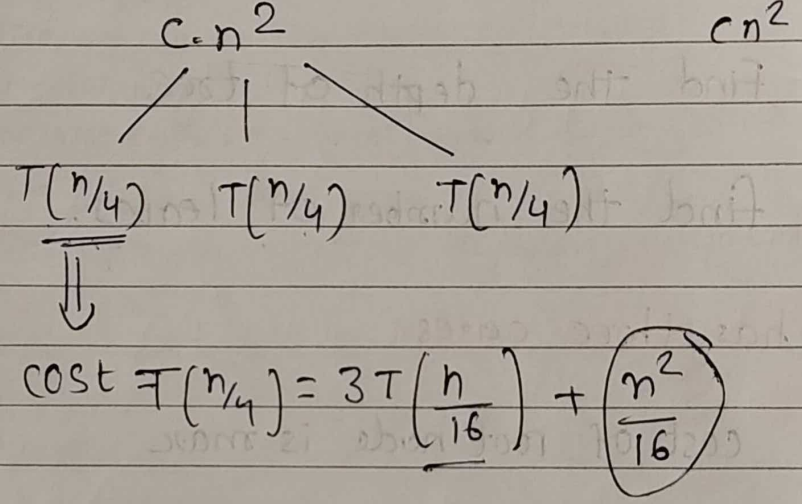


Case 1 : cost of ~~eac~~ Root Node is max

Example: $T(n) = 3T(n/4) + n^2$

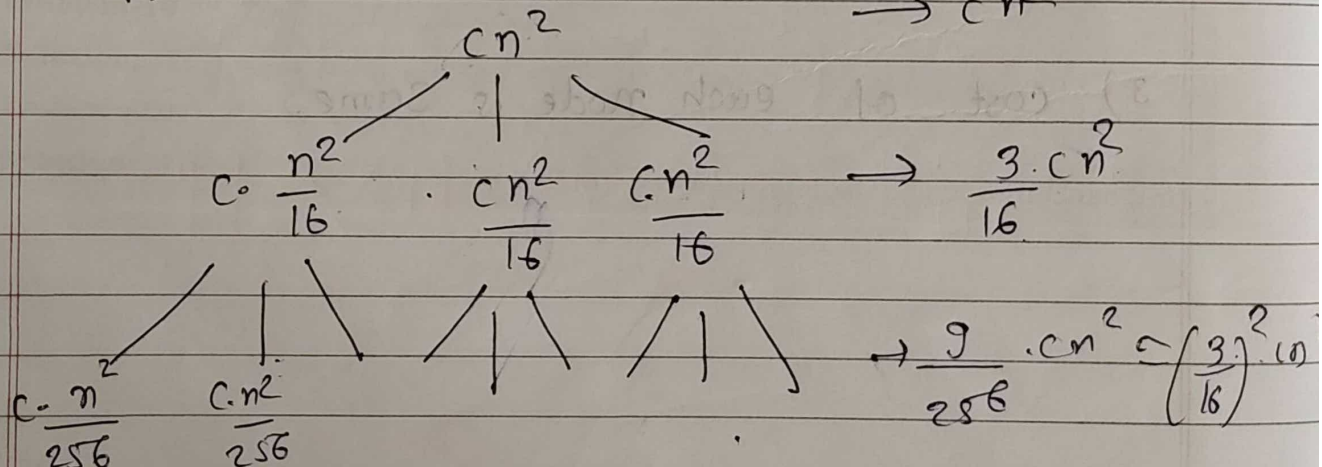
$T(n) = 3T(n/4) + n^2$ — root

Step: 1



$\text{cost } T(n/4) = 3T\left(\frac{n}{16}\right) + \frac{n^2}{16}$

Step: 2



$\text{cost } T(n/16) = 3T\left(\frac{n}{64}\right) + \frac{n^2}{256}\left(\frac{3}{16}\right)^3 cn^2$

cost of each level $= \left(\frac{3}{16}\right)^i c \cdot n^2$ → level

Step: 2

depth of tree =

$\frac{n}{4^i} = 1$

$T(n) = 3T(n/4) + n^2$

$\boxed{i = \log_4 n}$

totalcost = $T(n) = cn^2 + \frac{3}{16} cn^2 + \ldots + \left(\frac{3}{16}\right)^{d-1} cn^2 + n^{\log_4 3}$ (Till get 1) $+ n$

$= cn^2 \left\{ 1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \ldots \right\} + n^{\log_4 3} \times n^{\log_4 3}$

cost of leaf node $= \left(\frac{3}{16}\right)^i c \cdot n^2$ → depth

$= \left(\frac{3}{16}\right)^{\log_4 n} \cdot cn^2$

$= n^{\log_4 3}$

cost of leaf node

G.P series =

$a + ar + ar^2 + \ldots = \frac{a}{1-r}$, for $|r| < 1$

$= \frac{1}{1 - 3/16} = \frac{16}{13}$

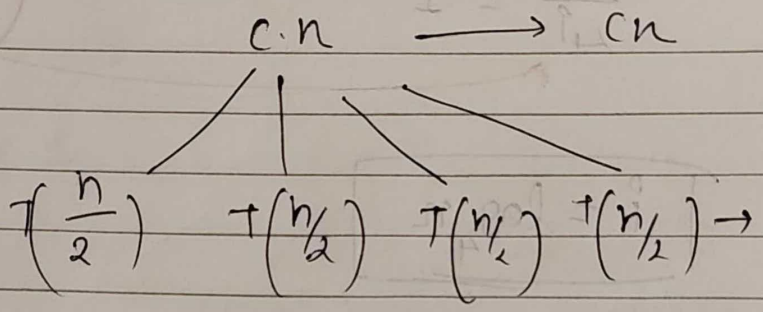
$\log_4 3 = 0.79$ $= c \cdot n^2 \left(\frac{16}{13}\right) + n^{\log_4 3}$

$= O(n^2)$

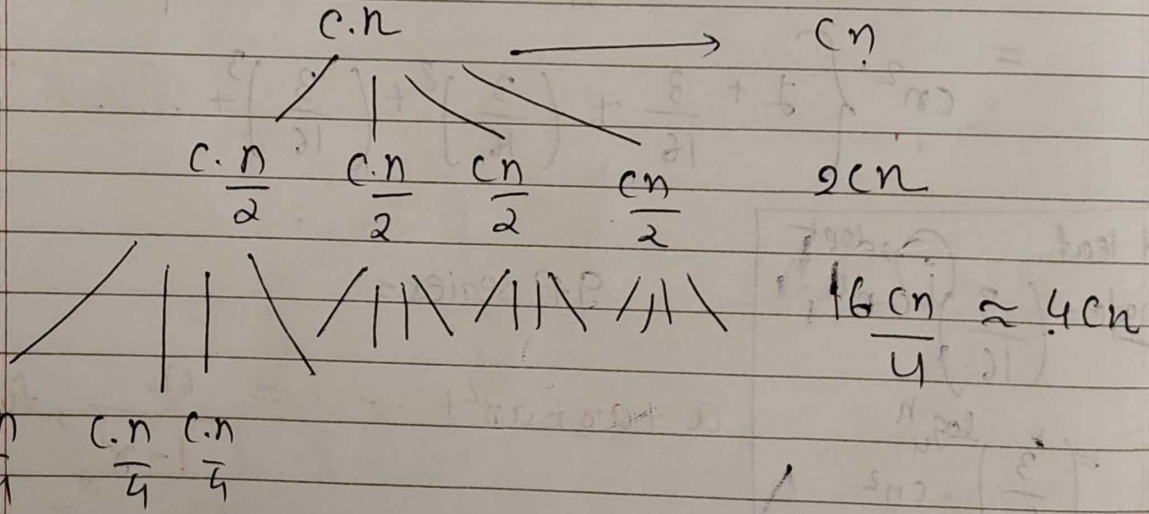
(2) cost of leaf node is maximum

$T(n) = 4T(n/2) + n$ — root

divide in 4 part



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



cost of each level $= (2)^i c.n$

dept of the tree $= \frac{n}{2^i} = 1$

$i = \log_2 n$

cost of leaf level node $= (2)^i c.n$

$= 2^{\log_2 n} . cn$

$= n^{\log_2 2} . cn$

$= c.n^2$

ii) no of leaves $= 4^i = 4^{\frac{\log n}{2}}$

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

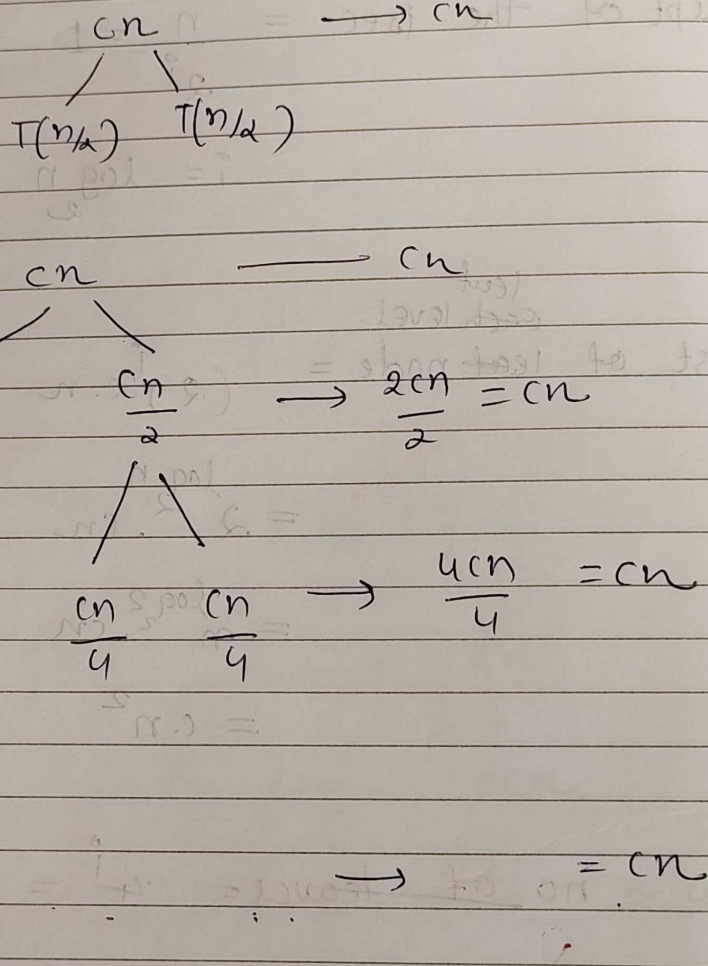
$= n^2$

Total cost $= cn + 2.cn + 8.cn + \ldots + n^2$

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

Case: 3 cost of each level is same

Q- $T(n) = 2T(n/2) + n$ (Merge sort)



Depth = $\frac{n}{2^i} = 1$

$i = \log_2 n$

Level = depth + 1

level = $\log_2 n + 1$

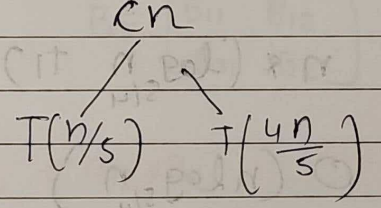
Total cost = cost of each level * No. of levels

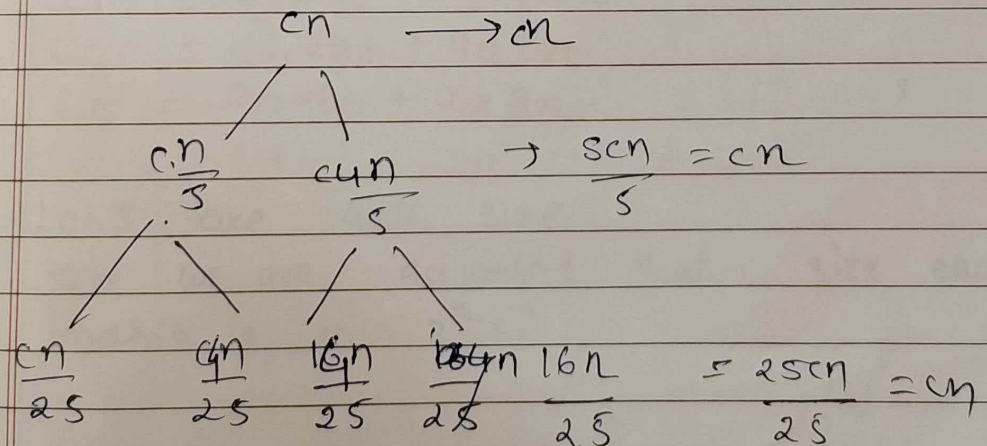
$= cn * (1 + \log_2 n)$

$= cn + cn \log_2 n$

Total $T(n) = \Theta(n \log_2 n)$

$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n$





cost of each level = n

for depth we have $T(n/5)$ and $T(4n/5)$

when we take $T(\frac{4n}{5})$, it will give more Depth

$$\text{Depth} = \frac{n}{(5/4)^i} = 1$$

$$i = \log_{5/4} n$$

$$\text{level} = \text{Depth} + 1$$

$$= (\log_{5/4} n + 1)$$

total cost = total cost of each level × no. of level

$$= n * (\log_{5/4} n + 1)$$

$$= O(n \log_{5/4} n)$$

* Divide and conquer

→ Matrix Multiplication

If we will use iterative algo then time complexity $O(n^3)$

can we achieve better ?

Yes, using ~~dei~~ devide & conquer

what will be smaller problem size ?
→ 2×2

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = a_{11} * B_{11} + a_{12} * B_{21}$$
$$C_{12} = a_{11} * B_{12} + a_{12} * B_{22}$$
$$C_{21} = a_{21} * B_{11} + a_{22} B_{21}$$
$$C_{22} = a_{21} * B_{12} + a_{22} B_{22}$$

Let's take 4×4 size
Here we are assuming that size each matrix is in $2^n \times 2^n$