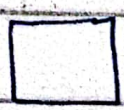
DESIGN AND ANALYSIS OF ALGORITHMS
BSE: 5th Semester

19/9/24

① Importance of Course:
(i) Topics
(ii) Content.

② Design (Structure) and Analysis (Correctness) of Algorithms (step by step procedure) (Sequence is important) (Independent of Programming language)

③ Computational Model (Computer):
Two core things of theoratical model
(i) Processor 
(ii) Memory 

④ Processor do computations (arithmetic operations, logical operations, Binary operations etc....).

⑤ Analysis (Correctness of Algorithm)
(Betterment of Algorithm according to parameter).

** Book: Introduction to Algorithms (3rd Edition)
(Thomas H Coreman)
: 4th edition is also available on internet.

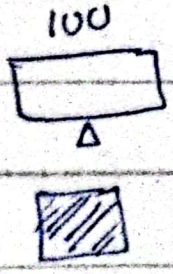
⊕ Better Parameters:
(i) Space (ii) Time (Number of steps)

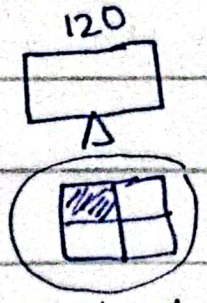
⊕ Basic Programming Languages / Algorithms:
(i) Individual Statement
(ii) selection
(iii) Repetition

⊕ Differentiate between algorithms on basis of number of steps / performance.

⊕ Execution time is of program.
(Depends on machine capability).

⊕





: Depends of dependency of steps.

(Parallel Algorithms) / Independent steps simultaneously work

Linear search

1 core	4 cores
100 steps	25 steps (parallel algorithm)

Theoretical	Real world
Time (number of steps).	execution time Machine dependency.

24/9/24

* Differentiating on two algorithms for same problem:
  (i) Time (ii) Space.
* Time means? Execution time will be of wall clock time. But Algorithm time will be number of steps.
* Algorithm will be at last be converted into code to run on machine.
* Multiple workers (core) can work better for some algorithms (But not for Bubble Sort).

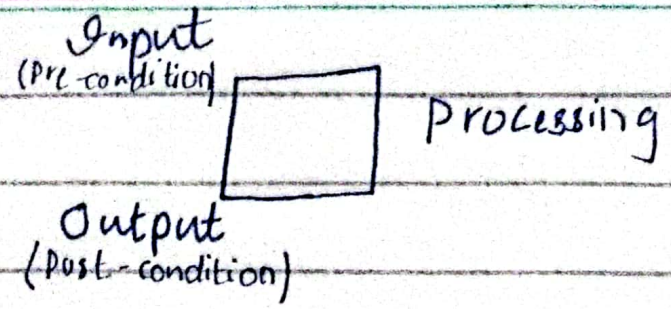
Better solution/Algorithm:

⇒ Analyze number of steps

* Every problem is supposed solved in algorithm in unit Time.
* BULK of program statements is in loop.

```
a ← b
a ← b+c
(Loop) ..... a < b
(More Time)
```

* Every Algorithm can take Input and do Processing and gives output.



⊛ Binary Search :

Input: Sorted Array, Key
Output: Index or -1

⊛ int arr[n], Key

```
int BinarySearch( int arr[n], x )
{
    low = 0;  high = n-1;
    mid = (low+high)/2;
    while ( low < high )
    { if (arr[mid] == x)
        { return mid;
        }
      else if ( arr[mid] < x)
      {   low = mid;
          mid=(low+high)/2;
      }
```

(*) A[n] , x ——— (1)

low = 0, high = n-1 ——— (1) (2)

while (low < high) → logn + 1

{ mid = ⌊(low + high)/2⌋ → logn (1)

if (A[mid] == x) → logn

{ return mid;

}

else if (A[mid] > x) → logn (1)

{ high = mid - 1;

}

else → logn

{ low = mid + 1;

}

}

end while;

return -1; ——— (1)

(*) 2 + [2 + 2 + 2 + 2 + ... 2] (?) (Times)

n n/2 n/4 n/8 1

n/2^0 n/2^1 n/2^2 n/2^3 ..... n/2^k = 1

n/2^k = 1 ⇒ n = 2^k (Taking log

log_2 n = log_2 2^k ⇒ log_2 n = k · log_2 2

$[2+2+\cdots+2]$ (2) (a) (3)

$2+2\log_2 n$

$K = \log_2 n$

size : 8

$2+2\log_2 8$

$2+2\log_2 2^3$

$2+6 = (8)$ Times

$[a+a+a+\cdots a]_{\log_2 n}$

$a \log_2 n$

$[a+a+a+\cdots a]_{10}$

$10\,a$

(*) $c = 0$ ——— (1)

for(i=0; i<n ; i++) — (n)

{ c++; }

$[1+1+1\cdots+1]_n$ (i: 0, 1, 2, ..., n-1)

(n+1)

c=0; ⟶ (1)

(*) for(i=0; i<n; i++) — (n)

{

for(j=0; j<n; j++) ⟹ $1+[n+n+n\cdots+n]_n$

c++;

}

$[1+1+1\cdots+1]_n$ (1n)

$[1+1+1+\cdots 1]_n$ (1n)

$1n * 1n = n^2$

$1+n^2$

c=0

(*) for(i=0; i<n; i=i+2)

{ for(j=0; j<n; j++) $1+[n+n+n\cdots+n]_{\frac{n}{2}}$

{ c++; }

}

$1+\frac{n^2}{2}$

$: \frac{n(n+1)}{2}$ $2a_1 + (n-1)d$

(*) c=0 ——————— (1)

```
for(i=n ; i>0 ; i--)
{
    for(j=0 ; j<i ; j++)
    {
        c++;
    }
}
```

bars (How this statement works)

[n, n-1, n-2, ---, 0]

n

(n)(n-1)(n-2)

[n+(n-1)+(n-2)+(n-3) + ---- +3+2+1+0]

i = n	+ n
i = n-1	+ n-1
⋮	⋮
i =	+ 0

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

(*) Sessional Activities (Quiz, Class Participation)

(Exam like Quiz) (7-8) (Alternative weeks)

(*) A.P : $T_n = a + (n-1)d$ $S_n = \frac{n}{2}[2a + (n-1)d]$

G.P : $T_n = a.r^{n-1}$ $S_n = a.\frac{1-r^n}{1-r}$ $S_\infty = \frac{a}{1-r}$

H.P : $T_n = \frac{1}{a + (n-1)d}$

∴ H.P is reciprocal of A.P

(Harmonic Progression) (Arithmetic Progression).

26/9/24

(*) A[n]

↓

Number of elements / Size

-> 2 + 2 log$_2$ n

constant variable.

-> 1 + n ? + n^2 ? + n^2/2

: ? (Any constant) ? + n(n+1)/2 ? + n^2/4

: It mainly depends on 'n' { Problem size / Input size / Data size }

(*) Some algorithm not depends on input size. (Coustant value). (Fixed steps).

(*) Mostly algorithm depends on input size. (These must impact functioning)

(*) Detailed number of steps:

```
c=0
for (i=0 ; i<n ; i++)
    c++ ;
```

: Every step counts

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

(Processing) CPU

(Memory Access) i, n

: These two times will be considered.

(*) Processing → clock Frequency (2.1 GHz) (cycles per sec).

: Suppose processor = 1 GHz

$$\text{Time} = \frac{1}{\text{freq}} = \frac{1}{10^9}$$

$$= 10^{-9} \text{ seconds} = 1 \text{ ns}$$

: 3 GHz (Modern) $= \frac{1}{3}$ ns $= 0.33$ ns

(*) Cache memory is bit fast.

Cache → RAM → CPU : Latency, Bandwidth
(nano) (micro) (milli)

(*)

$c_1 + c_2 \log_2 n$	$\log_2 n$
$c_1 + c_2 n$	$\sqrt{n}$
$c_1 + c_2 n^2$	n
$c_1 + c_2 \frac{n(n+1)}{2}$	$n \log_2 n$
$c_1 + \frac{n2}{c_2}$	n^2
	$n3$
	2^n
	$n!$

Increasing ↓

$\log_2 n$, $\sqrt{n}$, n, $n \log_2 n$, n^2, $n3$ — Polynomial

2^n, $n!$ — Exponential

(*) Table of some values of size.

Do these exercises

(+) If $\log_2 n$ solves for 1 sec, then other algorithm will solves for what time? (with respect to $\log_2 n$)

(*) If $\log_2 n$ is solved at 1GHz in 1 sec, what is problem size?

(*) Next Week Quiz.

Book Chapter # 1 (9-10 Pages) (Note Important points of sections and also exercise questions).

3/10/24

(*) If computational time is in min, sec, hours, it is acceptable otherwise it is days, months, years then it will not be practical (ofcourse)

(*) Quiz Discussion:

(1)

```
c=0
for(i=1; i<n; i=i+1)
{  for (j=1; j<(2*i); j=j+1)
   {  for (k=j; k>0; k=k-1)
      {   c=c+1;
      }
   }
}
```

④ c=0

```
for(i=1 ; i<n; i=i+1)        1, n, n-1 = 2n
{
    for(j=1 ; j<(2*i); j=j+1)   → 1, 2i, 2i-1 = ③
    {
        for(K=j ; K>0 ; K=K-1)   1, j+1, j      } 3j+2
        {                                       }
            c=c+1;                              j
        }
    }
}
```

(3j+2) (2i-1)

Assume:

$3j+2$

$= 3(1)+2$ ⑤

$j=1$

5	K=1
	1>0
	c=c+1
	K=0
	0>0

① K=j

```
while(K>0)
{   c=c+1;
    K=K-1;
}
```

$$\sum_{j=1}^{(2i-1)} (3j+2) = 3\sum_{j=1}^{(2i-1)} + 2\sum_{j=1}^{(2i-1)}$$

$$= 3\left[1+2+3+\ldots\ldots+(2i-1)\right] + 2\underbrace{\left[1+1+1+\ldots\ldots+1\right]}_{(2i-1)}$$

$$= 3\frac{(2i-1)(2i)}{2} + 2(2i-1)$$

$$= 3(2i-1)(i) + 2(2i-1)$$

$$= 6i^2 - 3i + 4i - 2 = 6i^2 + i - 2$$

$= 6i^2 + i - 2 + (4i)$: for $i = 1$

$= 6i^2 + 5i - 2$ (9)

(*) $(6i^2 + 5i - 2)_{(n-1)}$

$j = 1$

$1 < 2$

$k = 1$

$1 > 0$

$\sum_{i=1}^{n-1} (6i^2 + 5i - 2)$

$c = c + 1$

$k = 0$

$0 > 0$

$j = 2$

$2 < 2$

$\Rightarrow 6\sum_{i=1}^{n-1}(i^2) + 5\sum_{i=1}^{n-1} i - 2\sum_{i=1}^{n-1}$

$6(1 + 4 + 9 + 16 + \cdots + (n-1)^2) + 5(1 + 2 + 3 + \cdots + (n-1))$

$\Rightarrow \frac{6(n-1)(n-1+1)(2(n-1)+1)}{6} + \frac{5(n-1)(n-1+1)}{2} - 2(1+1+1+1)_{(n-1)}$

$-2(n-1)$: $\frac{n(n+1)(2n+1)}{6}$ (square series)

$\Rightarrow (n-1)(n)(2n-1) + 5\frac{(n-1)(n)}{2} - 2(n-1)$

$\div$ by 2

$\Rightarrow 2(n)(n-1)(2n-1) + 5(n-1)(n) - 4(n-1)$

$\Rightarrow (2n^2 - 2n)(2n-1) + (5n^2 - 5n) - 4n + 4$

$\Rightarrow (4n^3 - 2n^2 - 4n + 2n) + 5n^2 - 5n - 4n + 4$

$\Rightarrow 4n^3 - 2n^2 - 4n + 2n + 5n^2 - 5n - 4n + 4$

$\Rightarrow 4n^3 + 3n^2 - 11n + 4 + (2n) + (1)$

$\Rightarrow (4n^3 + 3n^2 - 9n + 5)$

: $C_1 n^3 + C_2 n^2 + C_3 n + C_4$

: Value(0,1) (Maybe incorrect)

(*) $C_1 n^3 + C_2 n^2 + C_3 n + C_4$

(Dominate value) (Eg: Adding other value of n^2, n will produce little effect)

($C_2 n^2 + C_3 n$) (They have some value) (But very low against dominate value)

C_4 → can be neglected

$\Rightarrow n^3$

(*) Rate of Growth (Performance of Algorithm)
Order of Growth

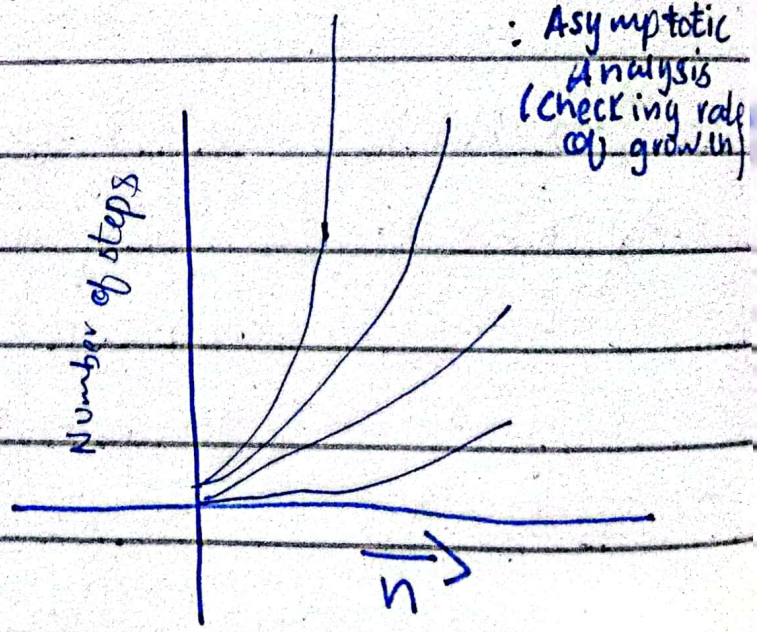
: We wanted to guage the performance of algorithm

⇒ Input size, Data size, Problem size is small then less effect.

⇒ When performance analysis we deal with larger input size.

(*) We study from graph:

⇒ log n
√n
n
n log n
n^2
n^3
2^n
n!
n^n



: Asymptotic Analysis (checking rate of growth)

8/10/24

* Asymptotic Notation:

↓

Analysis : Growth Rate

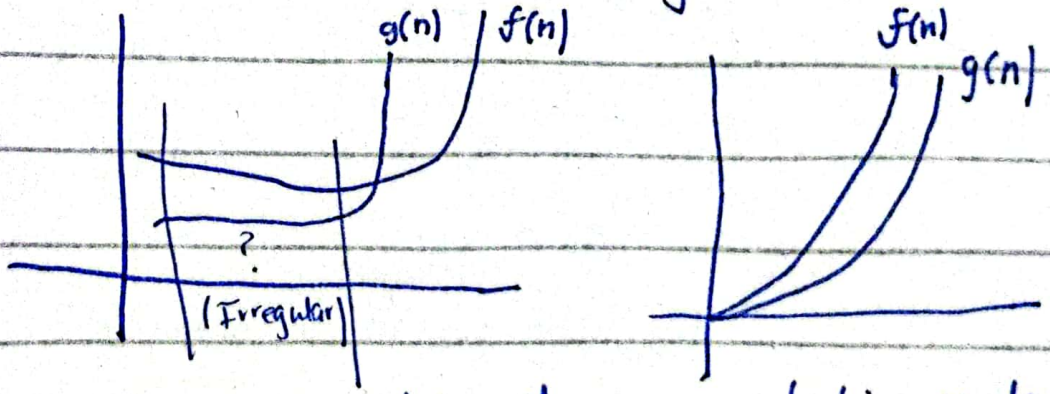
→ Running Time ∝ Number of steps

→ Related to growth

→ If N value large then how this algorithm will behave, so we do asymptotic analysis.
(For smaller value everything is fine but for large value we will study its behavior).

→ Scalability Analysis

⊛ Activity for increasing input size. (Excel)



(Here comes use of asymptotic analysis)

→ Similar curve to some other (Like like)

→ Categorization of class or families of function.

(1) Class of families of function:

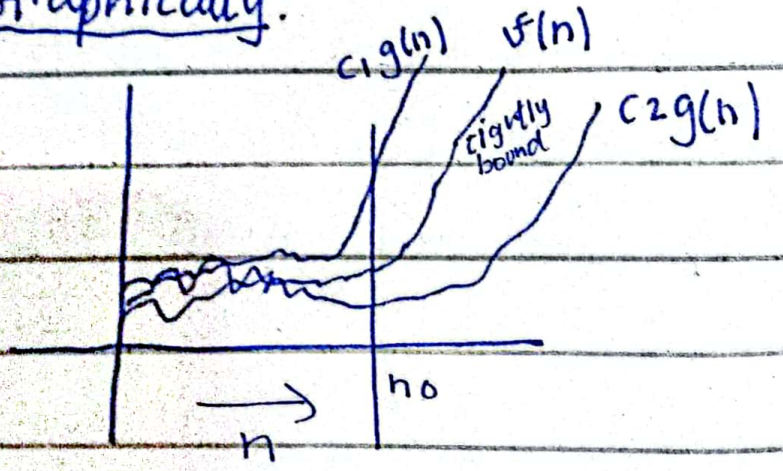
→ $3n^2 - 7n + 10$
$\frac{n^2}{3} - 8n + 12$
} $\equiv n^2$ (Behave as same as n^2)

⊙ Big-theta:

$f(n) \in \Theta(g(n))$
(set/class/family of function)

$= \{ g(n) \mid c_1, c_2 > 0, n_0$: tightly bound by family of two constants
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

: Graphically:



: At some time the function will not go out of family (boundaries)

$\therefore \Theta(g(n))$
(Tightly bound of function)

: n_0 (Input)

→ $\frac{n^2}{1000} + 1000n$
(Major Role till $1000 = n^2$)
(After $n^2 = 1000$ it behaviour will be like n^2)

: Lower order term becomes insignificant

(Its asymptotic analysis that after $n^2 = 1000$, it will not go outside of

→ Formal definition From book.
(f(n) and g(n) are interchanged).

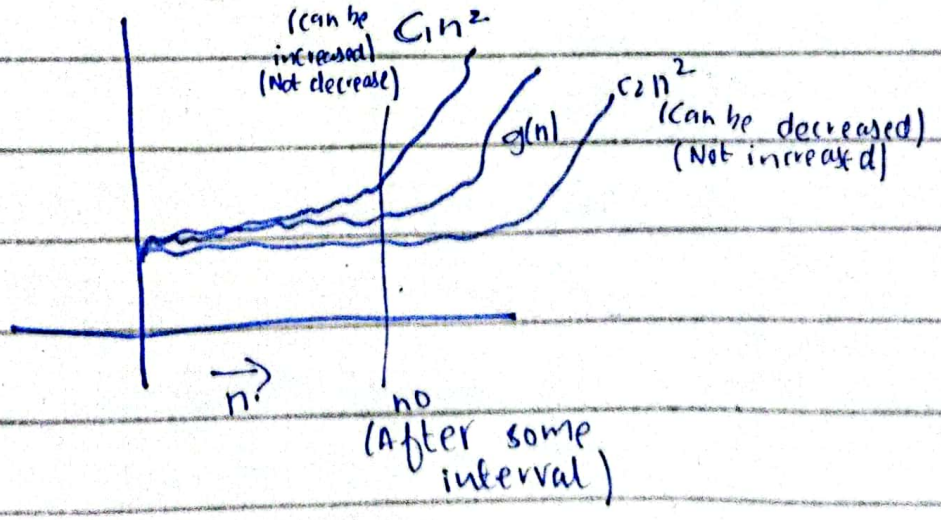
* $\Theta(f(n))$ is a set of functions.
(class / family / category)

same terminology as book:
$$\Theta(f(n)) = \{ g(n) \mid n_0, c_1, c_2 > 0,$$
$$0 \le c_1 f(n) \le g(n) \le c_2 f(n) \}$$

* $\Theta(n^2) = \{ n^2 \mid n_0, c_1, c_2 > 0,$
(May be incorrect) $0 \le \underbrace{c_1 f(n^2)}_{h^2} \le g(n^2) \le c_2 f(n^2) \}$

→ $\Theta(n)$ it about class/family/category of functions.

: $\Theta(n^2) = \{ g(n) \mid n_0, c_1, c_2 > 0,$
(Representing complete family)
(correct) $0 < c_1 n^2 \le g(n) \le c_2 n^2 \}$



$\rightarrow \Theta(n) = \{ g(n) \mid n_0, c_1, c_2 > 0$

$0 < c_1 n \leq g(n) \leq c_2 n \}$

$\rightarrow \Theta(f(n)) = \{ g(n) \mid n_0, c_1 c_2 > 0$

$0 < c_1 f(n) \leq g(n) \leq c_2 f(n) \}$

(*) Same terminology $\begin{cases} f(n) \in \Theta(n^2) \\ f(n) = \Theta(n^2) \end{cases}$: Abusing (Doing something unusual)

$f(n) = \frac{1}{2} n^2 - 3n \in \Theta(n^2)$

$\therefore \Theta(n^2) = \{ f(n) \mid n_0, c_1, c_2 > 0$

$0 < c_1 n^2 \leq f(n) \leq c_2 n^2 \}$

$\rightarrow$ other form:

$n_0 > 0$

$\therefore 0 < c_1 n^2 \leq \left(\frac{1}{2} n^2 - 3n \right) \leq c_2 n^2$: c_1, c_2 (Positive Constant)

$\rightarrow \frac{1}{2} n^2 - 3n \leq c_2 n^2$

$\frac{1}{2} n^2 - 3n - c_2 n^2 \leq 0$

$\therefore n^2 - 6n - 2c_2 n^2 \leq 0$

$n^2 - 2c_2 n^2 - 6n \leq 0$

$n^2 (1 - 2c_2) - 6n \leq 0$

: c_1 and c_2 can have multiple values because $f(x)$ can't go out others in graph.

: Inequality give range of values

$$n(n(1-2c_2)-6) \leq 0$$

$$n(1-2c_2)-6 \leq 0$$

$$n(1-2c_2) \leq 6$$

$$1-2c_2 \leq \frac{6}{n}$$

$$-2c_2 \leq \frac{6}{n} - 1$$

$$-2c_2 \leq \frac{6-n}{n}$$

$$\boxed{c_2 \geq \frac{6-n}{-2n}}$$

: $c_1 n^2 \leq \frac{1}{2}n^2 - 3n$

÷ by n^2

$$c_1 \leq \frac{1}{2} - \frac{3}{n}$$

$n=7$

$c_1 = \frac{1}{15}$

: for large value of n

: $\frac{1}{2}n^2 - 3n \leq c_2 n^2$

÷ by n^2

$$\frac{1}{2} - \frac{3}{n} \leq c_2$$

$n=1$

$c_2 = \frac{1}{2}$

no, $c_1 \geq 0$

So n will be true for both when equals to 7.

: $\boxed{n=7}$, $\boxed{c_1 = \frac{1}{15}}$, $\boxed{c_2 = \frac{1}{2}}$

(*) $3n^3 + 100n^2 \neq \Theta(n^2)$

(Proof by contradiction)

→ Proof:

$$\Theta(n^2) = \{ 3n^3 + 100n^2 \mid c_1, c_2, n_0$$

$$\therefore 0 < c_1 n^2 \leq 3n^3 + 100n^2 \leq c_2 n^2 \}$$

→ $c_1 n^2 \leq 3n^3 + 100n^2$

$\div$ by n^2

$c_1 \leq 3n + 100$

(It can be true)

$3n^3 + 100n^2 \leq c_2 n^2$

$\div$ by n^2

$3n + 100 \leq c_2$

(Cube term will never be less than square)

$\therefore n^3 \not\leq n^2$

(*) Homework:

: Big(O)(Upper Bound), Big(Ω)(Lower Bound)
(Not more than this) (Not less than this)

: small(O) and small(Ω)(Do it by yourself)

: Read from Book also.

15|10|24

(*) Every set (similar) of functions which have same growth rate for large value of n is called family of functions.

⊛ Theta Notation:

Large value of n / Behavior of Growth Rate.

→ This function will always not go above constant factor and not go below constant factor.

→ $f(n) = \frac{n^3}{100} - 100n^2 + \sqrt{n} - 10$

∴ (Instead of n^3, all terms could be dropped)

∴ $f(n) = \Theta(n^3)$

∴ f(n) not goes below constant factor n^3

∴ f(n) not grows more than constant factor n^3

($f(n)$ belongs to n^3) (Growth of $f(n)$ is same as n^3)

→ $0 \leq c_1 n^3 \leq f(n) \leq c_2 n^3 \quad \therefore \forall n \geq n_0$

→ Function growth rate not goes above constant factor or not goes below constant factor.

⊛ Big O:

→ Function growth rate not goes above constant factor.

× $O(g(n)) = \{ g(n) \mid n_0, c > 0$

$g(f_n) \leq c\, g(n)$

→ $O(g(n)) = \{ f(n) \mid n_0, c > 0$

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

$\}$

→ $O(n^2) = \{ f(n) \mid n_0, c > 0, \forall n > n_0$

(family name) (family function)

$0 \leq f(n) < c \cdot n^2$

$\}$

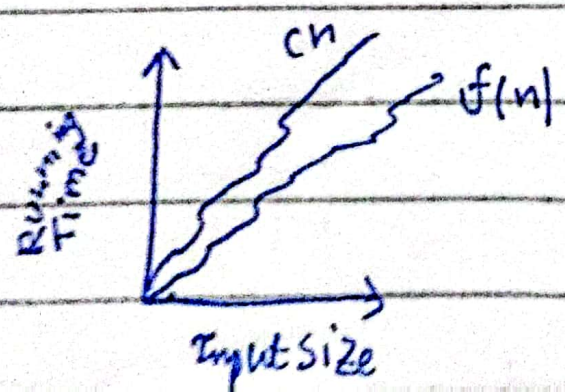
→ $O(f(n)) = \{ g(n) \mid n_0, c > 0$

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

$\}$

→ $f(n) = O(g(n))$: Not complete Notation Required

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



: Going up is bad Growing Rate

: Its worst case scenario

⊛ Big-omega:

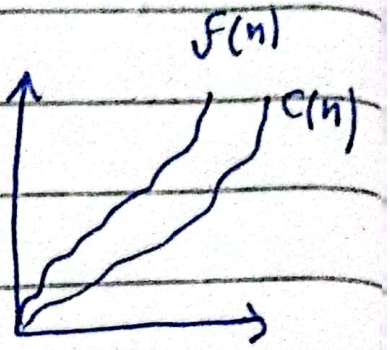
$\rightarrow \Omega(f(n)) = \{ g(n) \mid c > 0 \ \forall \ n > n_0$

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

$\}$

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

$0 \leq c.(n) \leq f(n)$



: Going down is good/best

: Growing rate is best possible value.

⊛ Little-Omega:

$\omega(f(n)) = \{ g(n) \mid c > 0 \ \forall \ n > n_0$

$0 \leq c.fn(n) < g(n)$

: Growth Rate will be more not equal

⊛ Little -o:

$o(f(n)) = \{ g(n) \mid c > 0 \ \forall \ n > n_0$

$0 \leq g(n) < c.f(n)$

: Growth Rate will be less not

→ $f(n) = 7n^3 + 1n^2 + 4$

$= (7n^3) + O(n^2)$: We are interested in growth rate.

$= \Theta(n^3)$: Not interested in precise value

→ The running time of Algorithm A is atleast $O(n^2)$: Worst case can't be atleast.
(Meaning less Statement)

→ If

$f(n) = O(n^2)$: Not perform worst then some other function

$f(n) = \Omega(n^2)$: Not perform best then some other function.

then

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

* (Iff (If and only if) question in Exercise)
* (We have to prove it by mathematical)

→ $f(n) = 2n^2 + 3n - 4$

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

$0 \leq f(n) \leq c \cdot n^2$

$$0 \leq 2n^2+3n-4 \leq c \cdot n^2$$

$\div$ by n^2

$$0 \leq 2+\frac{3}{n}-\frac{4}{n^2} \leq c$$

$$0 \leq 2+\frac{3}{n}-\frac{4}{n^2} \qquad \Big| \qquad 2+\frac{3}{n}-\frac{4}{n^2} \leq c$$

$n_0 = 1$

$c = 6$

: First calculate $n=1$

If $2+\frac{3}{n} = 5$ then $c=6$ (Always True)

$c=6$

$n_0=1$

$\Rightarrow f(n) = o(n^2)$

$\therefore 0 \leq f(n) < c \cdot n^2$

(Prepare for Quiz) $\therefore 0 \leq 2n^2+3n-4 < c \cdot n^2$

22/10/24

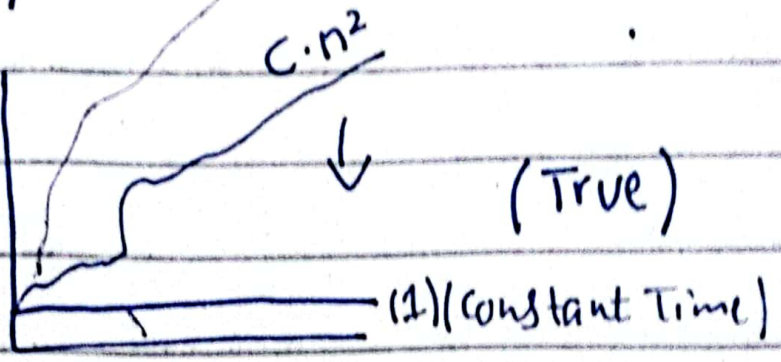
* Running Time / Time complexity is actually Asymptotic notation.
+ In general, we deal with Large values of n.
+ BigO(n2), not go above constant factor n2.

(*) n - Problem / Input / Data size.

(*) $\Theta(n)$ → Growth Rate not goes above or not go below constant factor.

(*) $c - \Omega(\lg n)$ → Running Time of Growth Rate constant factor not go below it.

(*) $\Theta(1) \rightsquigarrow O(n^2)$



(*) $O(n^2) \rightsquigarrow \Theta(n^2)$ (May or may not true) (May: Running time quadratic (May not: Running Time Linear or other)

$0 \leq f(n) \leq c_2 n^2$ $\forall n \geq n_{01}$

$0 \leq c_1 n^2 \leq f(n) \leq c_3 n^2$ (Now check for this) $\forall n \geq n_{02}$

: In General (Not True)

(*) $\Theta(1) \rightsquigarrow \Theta(n^2)$ (Not fulfill Lower bound state)

(*) Linear function running time can be true for $O(n^2)$ but not for $\Omega(n^2)$ (lower bound). $O(n) = O(n^2)$

(*) $\Omega(n) \rightsquigarrow O(n^2)$

24/10/24

* Algorithm A $f(n)$:

$f(n) = O(n)$ → , $f(n) = O(n^2)$

(True)

→ Because growth rate of n^2 is more than n, so, it will always be greater and the statement $f(n) = O(n)$ will always be true for $f(n) = O(n2)$.

→ $0 \leq f(n) \leq c \cdot n$ $(O(n))$

→ $0 \leq f(n) \leq c \cdot n^2$ $(O(n^2))$

$0 \leq f(n) \leq O(n) \cdot n$

* $f(n) = O(n)$

$0 \leq f(n) \leq c_1 n$

c_1 $\forall n \geq n_{01}$

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

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

$0 \leq f(n) \leq c_2 n^2$

c_2, $\forall n \geq n_{02}$

$c = \max(c_1, c_2)$ $0 \leq f(n) \leq cn^2$

$0 \leq f(n) \leq cn \leq cn^2$ and

$\therefore n$ is is a positive integer

→ if $cn \leq cn^2$ ÷ by cn

$1 \leq n$

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

$0 \leq f(n) \leq c_1 n^2$ $0 \leq f(n) \leq c_2 n$

$0 \leq f(n) \leq c_1 n^2 \leq c_2 n^2$

$c = \max(c_1, c_2)$

$0 \leq f(n) \leq c n^2 \leq cn$

Not possible for $n > 1$

if $cn^2 \leq cn$

÷ by cn

$n \leq 1$ (False) (Not Always True)

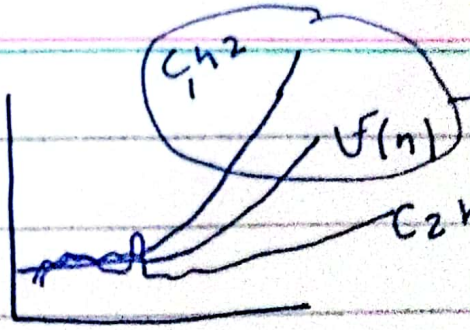
(*) $f(n) = O(n^2)$ ⇝ $f(n) = \Omega(n)$

$0 \leq f(n) \leq c_1 n^2$ $0 \leq c_1 \cdot n \leq f(n)$

$c = \max(c_1, c_2)$

$c \cdot n \leq f(n) \leq cn^2$

: Quiz (Tomorrow)



→ We can't comment from only this information

∴ $f(n)$ = Logarithmic function

(Contradiction Example)

→ Memory and Processor are important for every system.

→ Resource Sharing (Already prepared solution).

⊙ Divide and Conquer:

→ Recursion (Both go side by side).

→ How does it work?

(i) Divide problems into sub-problems of same kind. (Subproblems of the same nature)

(ii) Conquer: Solving them recursively. Straight forward Sol.

(iii) Parts:

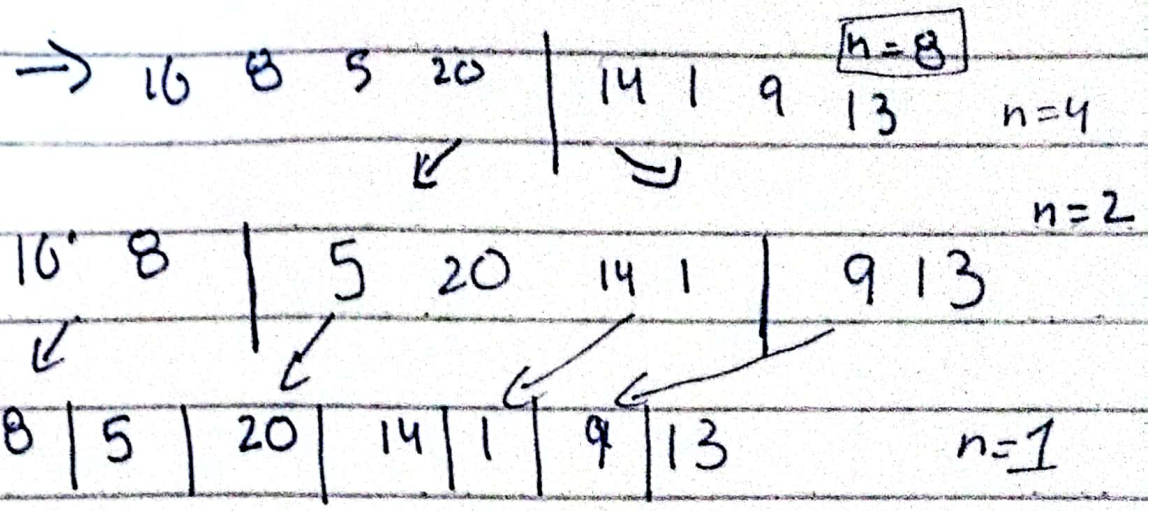
(i) Recursive Parts

(ii) Base case

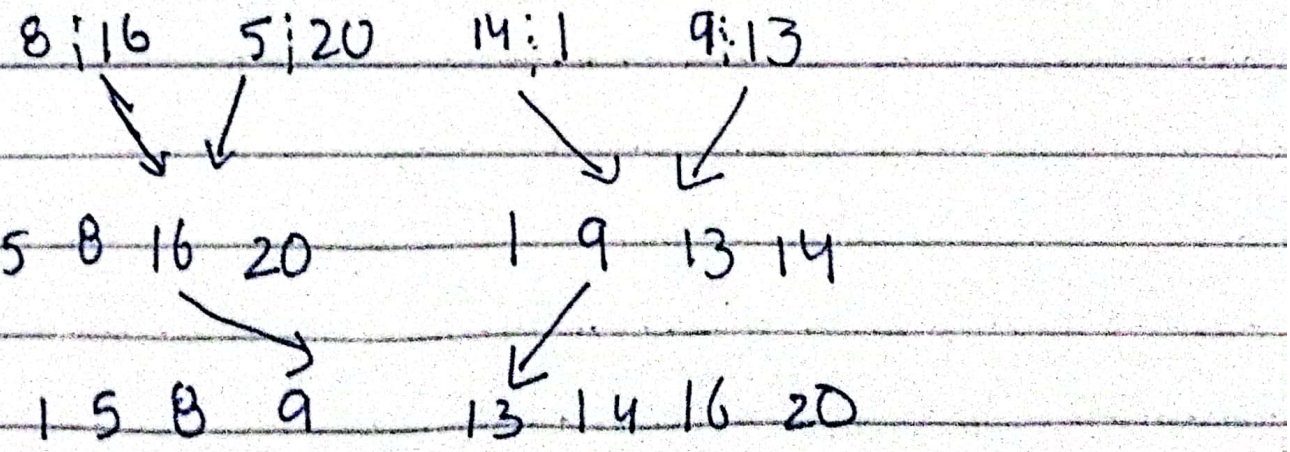
(iv) Divide, Conquer, Combine (Merged)

⊙ Merge Sort:

Procedure and method to sort the values in ascending and descending.

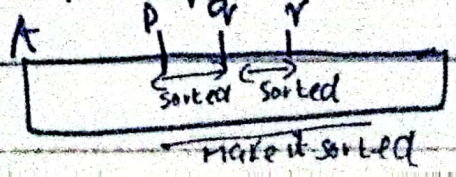


→ We will combine it.
(sorted array will be merged so that array will remain sorted)



⊛ Merge(A , p , q , r)

Array



n = 8

∴ n will be power of 2

29/10/24

⊕ It is not possible to solve every problem by Divide and conquer

⊕ Running Time of Algorithm: (Problem)

$$T(n) = \begin{cases} D & : n = c \\ aT\left(\frac{n}{b}\right) + \text{cost} & : n > c \end{cases}$$

(Sub problem) Combine

(Times reaches when we get straight forward solution.)

⇒ For Merge sort.

$$T(n) = \begin{cases} D(1) & \\ aT\left(\frac{n}{b}\right) + C & \\ T2T\left(\frac{n}{2}\right) + \Theta(n) & \end{cases}$$

⊙ Methods to solve recurrence relation:

1) Recursion Tree
2) Substitution Method
3) Master theorem

⊛ Binary Search Divide and Conquer.

→ First we target mid, then we apply recursion if it not solution.

→ Exercise for Binary Search using Divide and Conquer.

Write in simple word (Problem Statement) (Pseudo code)

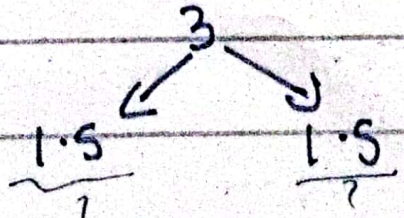
→ Recursive Algorithm, Recurrence Relation (T(n))

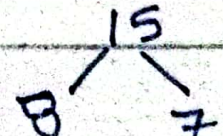
⊛ Linear Search by Divide and Conquer.

start, end

```
LinearSearch ( A[n] , n, Key)
{
    if(s!=e)
    :
    :
```

→ If n is odd, then it is not divided further equally in terms of 2. So we take 2^n to get problem size and division equally.





(So we apply ceil and floor)

(*) Must implement implementation of Binary and Linear Search.

(*) Factorial By divide and Conquer (Combine part is visible here)

(*) QuickSort (Also see it).

(*) Problem statement (Divide, Conquer), Recurrence Relation, Implementation.

(*) Page #88 4.4 (Methods) +Example. (Recursion Tree) (3-4 pages) (Exercise).

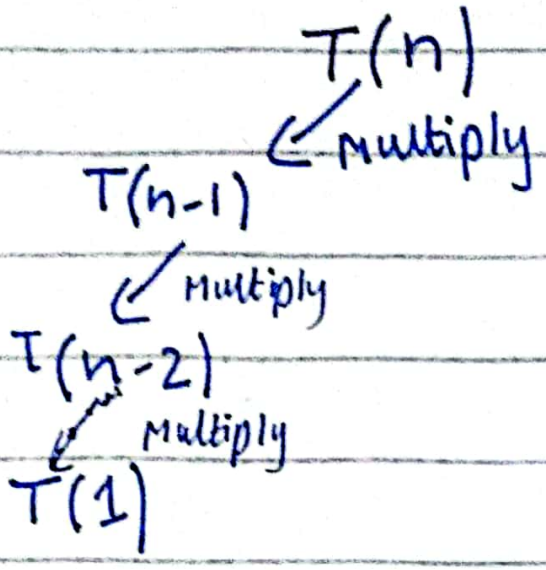
(*) (Linear Search(A[n], start, end, key)

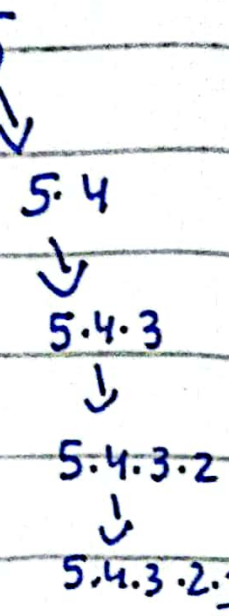
```
{
    if (start!=end)
    {
        if ( A[start] == key)
        {
            return true;
        }
        else
        {
            Linear Search( A[n], start+1, end, key)
        }
    }
    else
    {
        return false;
    }
}
```

if(s==end)
{ return false;
}
else if(A[s]==key)
{ return true;
}
else
{ LS(A[n],s+1,e,key);
}

⊙ Tree:

Factorial





⊛ $T(n)$

"Time to solve problem of size n"

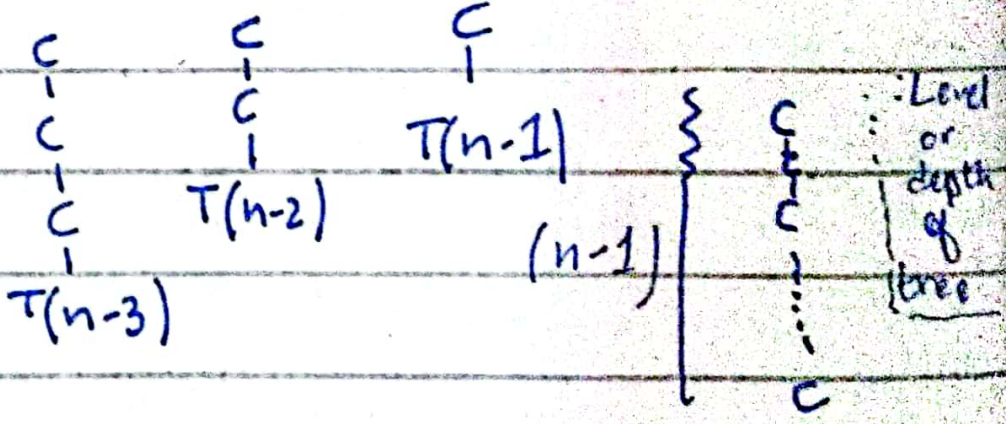
: Multiplicative factor decrease (2, 4, 6, 8, 10...)

: Additive factor decrease (1)

$$\Rightarrow Tn = \begin{cases} \Theta(1) \\ T(n-1) + \Theta(1) \end{cases}$$

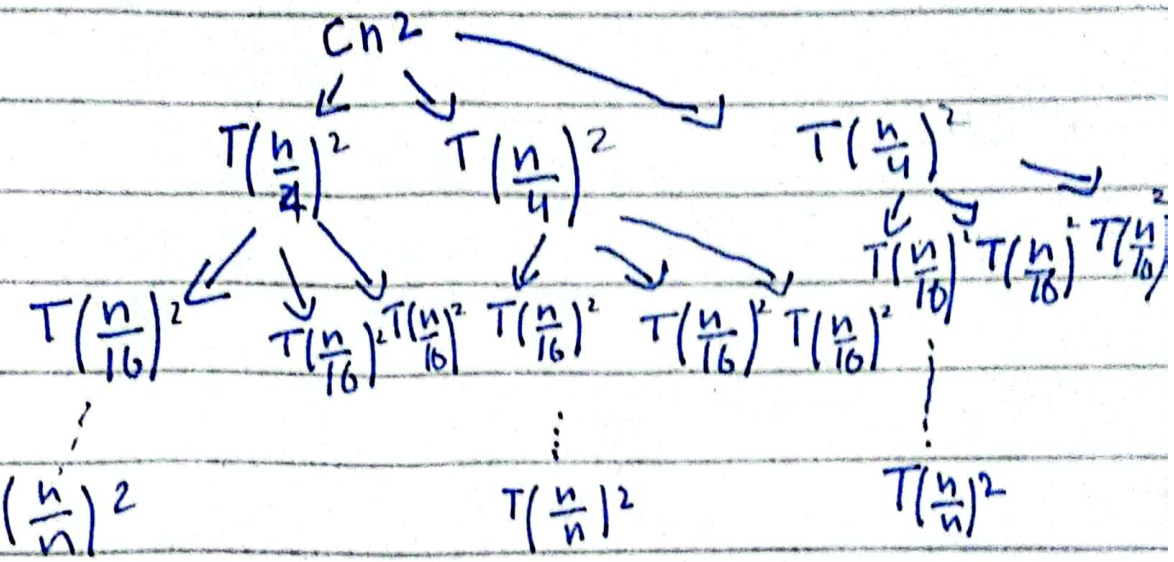
$$\Rightarrow Tn = T(n-c) + c$$

⇒ Tree:



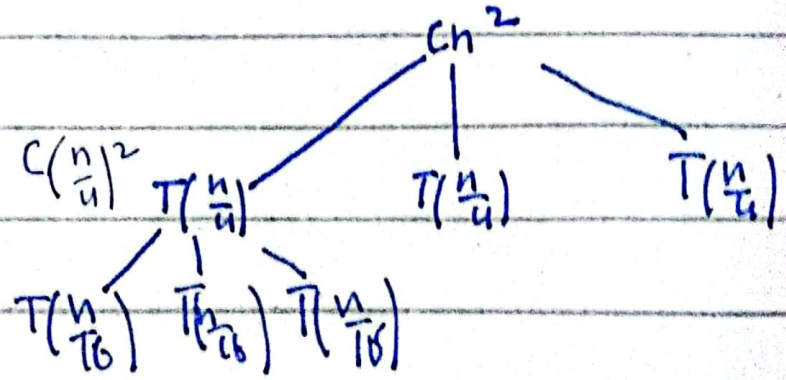
$$T(n) = \overset{(n-1)c}{c(n-1)}$$
$$= cn - c$$
$$= \Theta(n)$$

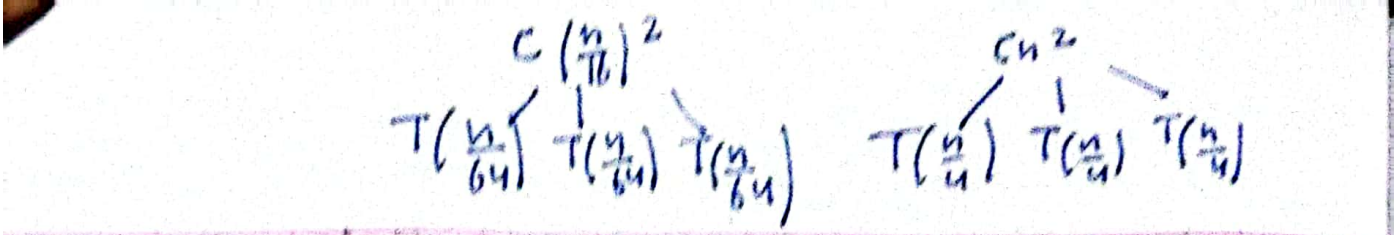
(*) $T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$

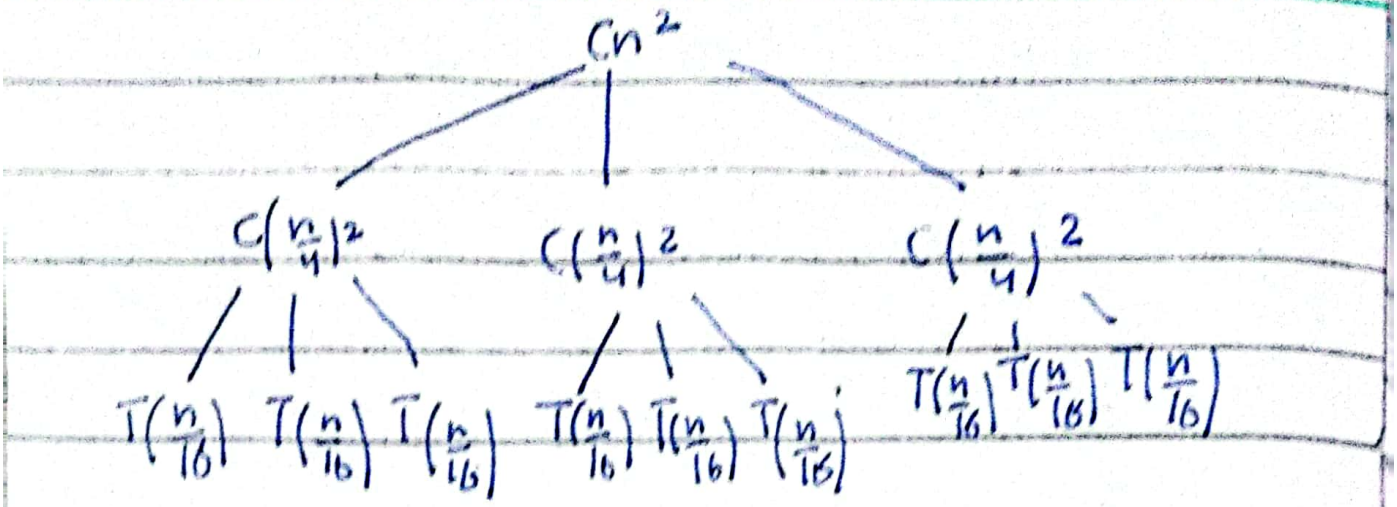


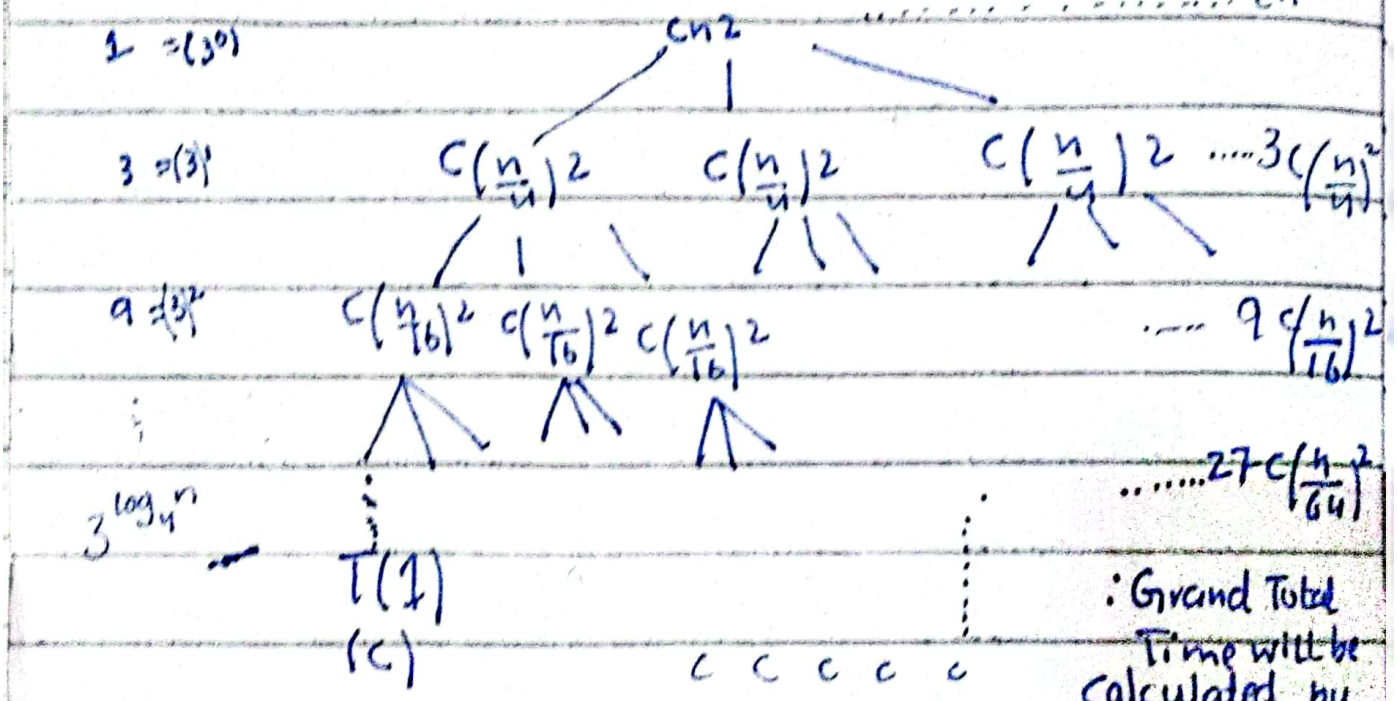
$\Rightarrow T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$

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









: Grand Total Time will be calculated by adding each value.
: Horizontal sum

$\Rightarrow 0 - cn^2 = cn^2\, 3^0 (\frac{1}{16})^0$

$1 - 3c(\frac{n}{4})^2 = cn^2\, 3^1 (\frac{1}{16})^1$

$2 - 9c(\frac{n}{16})^2 = cn^2\, 3^2 (\frac{1}{16})^2$

$3 - 27c(\frac{n}{64})^2$

$\vdots$

$\log_4 n \quad cn^2 (\frac{3}{16})^{\log_4 n}$

→ $n, \frac{n}{4}, \frac{n}{16} \ldots\ldots n$

$\frac{n}{4^0}, \frac{n}{4^1}, \frac{n}{4^2} \ldots\ldots \frac{n}{4^k} = 1$

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

$n = 4^k$

$n = 4^k$

$\log_4 n = \log_4 4^k$

$k = \log_4 n$

→ $3^{\log_4 n}$

$n^{\log_4 3}$

$i = ?$ $(0, 1, 2, 3 \ldots \log_4 n)$

$c n \log_4 3$

5/11/24

Quiz on Wednesday

Substitution Method:

→ Recurrence relation is found, now how would it behave (its growth rate) (Asymptotic bound).

→ By observing the recurrence relationship structure, we can guess the asymptotic bound (mainly upper bound (Big O) and lower bound (Big Ω)).

→ This method tells the whether the guessed bound is correct or not (may be overestimated (tight bounds) or underestimated, so we adjust)

→ This method is to verify some relation.

(Two methods) → ① Derive some relation and verify
② Guess some relation and verify.

→ **Parts:**
① Guess the form of solution
② Prove the guess using mathematical induction.

⊙ **Proof by Induction:**
→ First we solve for base case.
→ Then we use Inductive step (for lower limits it is true)

(* Steps)
→ First for 1, K, K+1
→ Base Case, Lower Limits / Inductive hypothesis), Upper Limits / Inductive step)

⊙ **E.g:**

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

⊙ Guess:

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

$$\therefore T(n) \leq c.n\log n$$

: No method for guessing
: So we assume by observing

① Inductive Hypothesis:

: we are assuming it is true for all lower values so it will be true for $\frac{n}{2}$

$$m < n$$

$$k < n$$

$$T\left(\frac{n}{2}\right) \leq c\frac{n}{2}\log\frac{n}{2}$$

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

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

$$\leq 2\left[c\frac{n}{2}\log\frac{n}{2}\right] + n$$

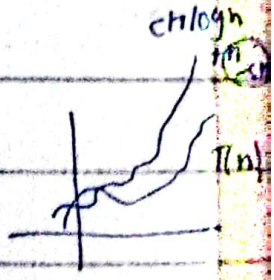
$$= cn\left[\log\frac{n}{2}\right] + n$$

$$= cn\left[\log n - \log 2\right] + n$$

$$= cn\left[\log n - 1\right] + n$$

$$= cn\log n - cn + n$$

$$= \boxed{cn\log n + (n - cn)}$$



: $cn\log n + (n - cn) \leq cn\log n$

$$n - cn \leq 0$$

: for all values of $c > 0$

$$\leq cn\log n \quad : \text{(More small value)}$$

$$T(n) \leq cn\log n$$

: $(n - cn)$ less than or equal zero (so negative)

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

→ $T(n) = O(n \log n)$

$T(n) = O(n^2)$ (will be true for it)

$T(n) = O(n)$

Other guess:

→ Guess:

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

$$T(n) \leq c \cdot n^2$$

→ Inductive Hypothesis:

$$m < n^2$$

$$k < n^2$$

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2$$

$$T\left(\frac{n}{2}\right) \leq c\frac{n^2}{4}$$

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

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

$$= \frac{cn^2}{2} + n$$

$$\frac{cn^2}{2} + n \leq c \cdot n^2$$

$$n \leq cn^2 - \frac{cn^2}{2} \Rightarrow n \leq \frac{cn^2}{2}$$

$\frac{1}{n} \leq \frac{c}{2}$

$n \leq \frac{2}{c}$

$nc - 2 \leq 0$

: Base case is not discussed yet

: Check it from book (Base case is easy)

: Master theorem use

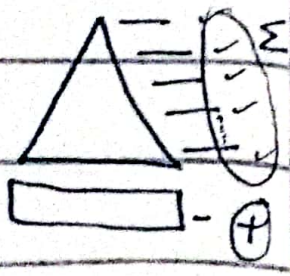
7/11/24

→ Maximum- Sub Array Algorithm use

⊙ Master Theorem:

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

$\Rightarrow n^{\log_b a}$ $\quad f(n)$



Three cases on basis of growth rate comparison.

⇒ less, equal, greater relation

⇒ Time will be $f(n)$, $n^{\log_b a}$ or sometimes $n^{\log_b a}$

⇒ $\Theta(f(n))$, $\Theta(n^{\log_b a})$, $\Theta(n^{\log_b a} \cdot \log n)$

① if $(n^{\log_b a}) > f(n)$ $\quad \Theta(n^{\log_b a})$

② if $(n^{\log_b a}) = f(n)$ $\quad \Theta(n^{\log_a b} \cdot \log n)$

③ if $(f(n) > n^{\log_b a})$ $\quad \Theta(f(n))$

(+) In recursion tree we mention combine cost and adding up leaf nodes cost and overall sum of G.P and leaf node value.