



Abdul Bari Lecture Notes

* Algorithms:

Step-by-step procedure for solving a problem,
often Computational Problems?

- Difference between 'program' & 'algo':

Algorithm Program

- i) Design Phase
 - ii) Requires Domain Knowledge
 - iii) Written in any language
(mathematical notations & Pseudocode recommended)
 - iv) Not dependent on HW or OS
 - v) We analyse algorithms
 - i) Implementation phase
 - ii) Programming Knowledge
 - iii) Programming Languages
(C, C++, Java, Python)
 - iv) Dependent
 - v) We test programs

Priori Analysis &

- i) Analysis of alg
(Time, Space, ...)

- (Time, space, ...)
 - ii) Language Independent
 - iii) H/W Independent
 - iv) Not actual time
but the time
function

• Posteriori Testing

- ## i) Testing of programs (Bytes, seconds, ...)

- ## ii) Dependent

- ### iii) Dependent

- iv) Actual figures of time & storage required.



Date: _____

• Characteristics of Algorithms:

- i) Input can be 0 or more
- ii) Output at least 1 thing (serve a purpose)
- iii) Definiteness, every statement should be solvable and do-able, it should not be vague or ambitious
- iv) Finiteness, algo should terminate
- v) Effectiveness, should complete task efficiently, should not make unnecessary steps.

• Pseudocode Form of writing algorithms:

i) Algorithm swap(a, b) { ii) Algo Swap(a, b). begin
 temp = a ; temp $\leftarrow a$;
 $a = b$; a $\leftarrow b$;
 $b = temp$; b $\leftarrow temp$;
}

end

Syntax does not matter, should be understandable

• Analyzing algorithms: (Criteria)

1. Time - Getting time function
2. Space - Getting space function
3. Data Transfer
4. Power Consumption
5. CPU Registers (Device drivers, embedded)



Date: _____

Eg. of algo analysis: (Small Overview)

Algorithm Swap(a, b)

(1 § 1) ~~Assumptions~~ We assume that
simple statements
take constant '1' time

temp = a;

a = b;

b = temp;

}

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

$$f(n) \rightarrow O(1)$$

No matter the
no. of operations

eg: Even $x = a * 5 + b \dots (1)$

However, we can also do in-depth analysis and
not assume such statements constant, analysing all
the way upto machine code

Real World eg; i) Plan to go to friend's house
ii) Plan to successfully land on Mars.

Similarly, for space, each variable used is assumed
to use constant space

a - 1

b - 1

$$\text{temp} - 1 \quad S(n) = 1 + 1 + 1 = 3$$

$$S(n) \rightarrow O(1)$$

★ Frequency Count Method: (IMP)

Used to find Time & Space Complexity of
Algorithms by analyzing the accurate
Pseudocodes



Date: _____

Eg:

i) Algo Sum(A, n) {
 s = 0; 1
 for (i = 0; i < n; i++) { (n+1)
 s = s + A[i]; (n)
 }
 return s; 1
}

$$\therefore f(n) = n+1+n+1+1 = 3n+3 \rightarrow O(n)$$

$\therefore S(n) : \left. \begin{array}{l} s..1, & \& A..n \\ i..1, & & n..1 \end{array} \right\} \therefore S(n) = 1+1+n+1 = n+3 \rightarrow O(n)$
Auxiliary Space
Complete Space

Note: We ignore constant & less dominant terms to find complexity bounds.

ii) Algo Multiply (A, B, n) { . . . where A & B are Matrices
 for (i = 0; i < n; i++) { (n+1)
 for (j = 0; j < n; j++) { (n(n+1))
 c[i, j] = 0; n²
 for (k = 0; k < n; k++) n²(n+1)
 }
 }
 c[i, j] = c[i, j] + A[i, k] * B[k, j]; n³
}

$\therefore f(n) = (n+1) + n^2 + n + n^2 + n^3 + n^2 + n^3 = 2n^3 + 3n^2 + 7n + 1$

$\therefore S(n) : \left. \begin{array}{l} A..n^2, i..1, C..n^2 \\ B..n^2, j..1 \\ n..1, k..1 \end{array} \right\} \rightarrow O(n^3)$
 $\therefore S(n) = 3n^2 + 4 \rightarrow O(n^2)$



Date: _____

- Analysis of different iterative algorithms using 'Frequency Count Method': ... Understanding Approach

[Only the important & unique ones are mentioned here.
for more, watch video.]

i	j	no. of times
0	0*	0
1	0*	1
1	1*	1
2	0*	2
2	1*	1
2	2*	1
n		

$$\therefore \text{Total No. of times} = 0 + 1 + 2 + \dots + n \\ = n(n+1)/2$$

$$\therefore f(n) = (n^2 + n)/2 \\ \rightarrow O(n^2)$$

i	p
1	0
2	1 = 1+0
3	3 = 1+2
4	6 = 1+2+3
5	10 = 1+2+3+4
⋮	⋮
k	$\sum_{i=1}^{k-1} i \approx \frac{k(k+1)}{2}$

∴ Break point: $p > n$

Assume break at 'k'

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

$$\therefore \frac{k^2+k}{2} > n$$

Remove const. & less dominant,

$$\therefore k^2 > n$$

Equalize & take root, i.e. $k = \sqrt{n}$

$$\therefore f(n) \Rightarrow O(\sqrt{n})$$



Date: _____

③ $\text{for } (i=n; i \geq 1; i = \frac{i}{2}) \{$

stmt;	i	n
}	\vdots	$n/2$

Break at: $i \leq 1$

$\therefore n/2^k \leq 1$

$\therefore n \leq 2^k$

$\therefore \log n \leq k$

$\therefore O(k) = O(\log n)$

④ $\text{for } (i=0; i*i < n; i++) \{$

stmt;	i	$i*i$
}	0	0
	1	1^2
	2	2^2
	3	3^2
	\vdots	\vdots

Break at: $i*i \geq n$

$k^2 \geq n$

$k \geq \sqrt{n}$

$\therefore O(k) = O(\sqrt{n})$

⑤ Using While Loop:

$a=1;$

$\text{while } (a < b) \{$

stmt;	a
}	1
	2
	2^2
	2^3
	2^4
	2^5

Break: $a \geq b$

$2^k \geq b$

$$\therefore k \geq \log_2 b$$

$$\therefore O(k) = O(\log b)$$

Considering ' n ' as common notation we get $O(\log n)$



Date: _____

(6) ~~i = 1;~~

k = 1;

while ($k < n$) {

stmn;

k = k + i;

i++;

}

Break at: $k \geq n$

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

$$m^2 \geq n, m \geq \sqrt{n}$$

$$\therefore O(m) = O(\sqrt{n})$$

(7) while ($m \neq n$) { if ($m > n$)

m = m - n;

else

n = n - m;

}

Best Case: $m = 2, n = 2 \dots 1 \text{ time}$

Avg Case / Worst Case:

 $m = 16, n = 2$

16 : 2

14 : 2

12 : 2

10 : 2

8 : 2

6 : 2

4 : 2

2 : 2

General
notation
 $O(n/2)$ $O(n)$



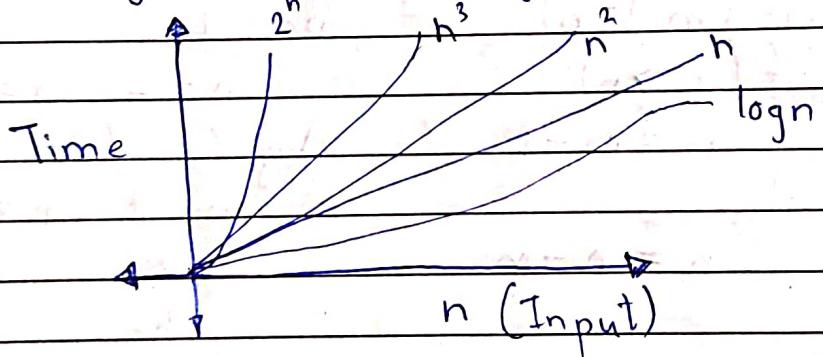
Date: _____

- Types of Time functions:

$O(1)$	constant (eg: $O(1000), O(50) \dots$)
$O(\log n)$	Logarithmic, $O(\sqrt{n})$ root
$O(n)$	Linear (eg: $O(2n+3), O(145n+24) \dots$)
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential
$O(3^n)$	
$O(n^r)$	

- Comparison of Time functions: (Always true for larger values)

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



- * Asymptotic Notation: (from Mathematics)

i) Big-Oh, (O) gives upper bound

$f(n) = O(g(n))$ if \exists +ve constant 'c' and ' n_0 ' such that

$$f(n) \leq c * g(n) \quad \forall n \geq n_0$$

e.g: $f(n) = 2n+3$, Let's take c as 10, n_0 as 1
 $\therefore 2n+3 \leq 10n, n \geq 1$



Date: _____

Here, $f(n) = 2n + 3$, $c = 10$, $g(n) = n$, $n_0 = 1$
We can also take,

$c = 1000$ or 100 or 7 till, $c \cdot g(n) \geq f(n)$

To take closest (tight) c :

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

$$2n + 3 \leq 5n \quad \text{for } n \geq 1$$

$$\therefore f(n) = O(n) \rightarrow \text{Most useful}$$

We can also write:

$$2n + 3 \leq 5n^2 \dots \text{True}$$

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

Here $f(n)$ belongs to class $O(n)$ thus all functions that grow faster ($O(n^2), O(n^3), \dots, O(n^n)$) are suited to be the 'upper-bound'. [$f(n) = O(2^n), f(n) = O(n^3)$] All functions with lesser growth ($O(1), O(\log n), O(\sqrt{n})$) are thus 'lower-bound'. [$f(n) \neq O(\log n), f(n) \neq O(1)$] And 'n' i.e $O(n)$ itself is the 'Average-bound'.

ii) Big-Omega, (Ω) gives lower bound

$f(n) = c \cdot g(n)$ if \exists +ve constant 'c' and 'no' such that

$$f(n) \geq c * g(n) \quad \forall n \geq 1$$

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

$$2n + 3 \geq 1_n \quad \forall n \geq 1$$

$$f(n) \geq g(n)$$

Similar to upper-bound properties we can write, here,

Most $\Leftrightarrow [f(n) = \Omega(n)]$, $f(n) = \Omega(\log n)$, $f(n) = \Omega(1)$
useful $\Leftrightarrow [f(n) \neq \Omega(n^2)]$, $f(n) \neq \Omega(2^n)$



Date: _____

iii) Big-Theta, (Θ), most accurate, gives avg. bound

$f(n) = \Theta(g(n))$ if \exists +ve constant 'c₁' and 'n₀', c₂' such that

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

Here, multiple classes cannot hold true!

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

$$\frac{1}{c_1} n \leq 2n+3 \leq \frac{5}{c_2} n$$

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

$f(n) = \Theta(n)$... Exact function

* Important: These terms are not related to best or worst case!

More Examples:

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

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

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

ii) $f(n) = n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$
 $= 1 \times 2 \times 3 \times \dots \times n$

$$1 \times 1 \times 1 \leq 1 \times 2 \times 3 \times \dots \times n \leq n \times n \times n \times \dots \times n$$
$$1 \leq n! \leq n^n$$

Here, we cannot have same thing on both sides.
 $\therefore \Theta \rightarrow$ not possible, $f(n) = O(n^n)$, $f(n) = \Omega(1)$



Date: _____

Thus, as Θ gives tight bound that is closest to the actual function, it is preferred over ' O ' & ' Ω '. If not possible, we then consider ' O ' & ' Ω '.

iii) $f(n) = \log n!$

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

$$O \approx 1 \leq \log n! < \log n^n \approx n \log n$$

$$\therefore f(n) = O(n \log n), f(n) = \Omega(1)$$

• Properties of Asymptotic Notations:

i) If $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$

eg: $f(n) = 2n^2 + 5 \rightarrow O(n^2)$

$$7 \cdot f(n) = [4n^2 + 35] \rightarrow O(n^2)$$

Also applies for $\Omega(g(n))$ & $\Theta(g(n))$

ii) If $f(n)$ is given then, $f(n)$ is $O(f(n))$

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

iii) If $f(n)$ is $O(g(n))$ & $g(n)$ is $O(h(n))$, $f(n) = O(h(n))$

eg: $f(n) = n$, $g(n) = n^2$, $h(n) = n^3$

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

$$g(n) = O(h(n)) = O(n^3)$$

then $f(n) = O(n^3)$



Date: _____

iv) If $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$ eg: $f(n) = \Theta(g(n))$, $f(n) = n^2 \Rightarrow g(n) = n^2$ Thus, $f(n) = \Theta(n^2)$ $g(n) = \Theta(n^2)$!! Only true for ' Θ ' notation.v) If $f(n) = O(g(n))$ then $g(n)$ is $\Omega(f(n))$!! True for ' O ' & ' Ω '.eg: $f(n) = n$, $g(n) = n^2$ then n has $O(n^2)$ & n^2 has ~~$\Omega(n)$~~ vi) If $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$ then, $f(n) = \Theta(g(n))$ *: Uncommon log property: $a^{\log_b c} = b^{\log_a c}$

* Comparison of Functions:

1. Input values and check

2. Use log and check coeff. of bigger terms

Q. $f(n) = n^2 \log n$, $g(n) = n(\log n)^{10}$

Now, $\log[f(n)] = \log(n^2 \log n) = \log n^2 + \log(\log n)$
 $= 2 \log n + \log(\log n)$..(i)

$\log[g(n)] = \log(n(\log n)^{10}) = \log n + \log(\log n)^{10}$
 $= \log n + 10 \log(\log n)$..(ii)

Compare ' $\log n$ ' in (i) & (ii) $\therefore 2 \log n > \log n$ (Asymptotically equal)

$\therefore f(n) > g(n)$



Date: _____

Q. $f(n) = 3n^{\sqrt{n}}$ $g(n) = 2^{\sqrt{n} \log_2 n}$

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

As, $3n^{\sqrt{n}} > n^{\sqrt{n}}$ (Asymptotically equal)
 $\therefore f(n) > g(n)$

* Best, Worst & Average Case Analysis:

1. Linear Search

Best Case: Target at first index $\rightarrow O(1)$

Worst Case: Last Index or not present $\rightarrow O(n)$

Average Case: All possible case times = $\frac{1+2+3+\dots+n}{n}$

Not possible
for every algo.

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

$\rightarrow O(n)$

* Again, Cases are not related to notations!

Notations are for function bounds.

Cases are type of analysis done on algo

2. For Linear Search:

Best Case: $B(n) = 1 = O(1) = \Omega(1) = \Theta(1)$

Worst Case: $W(n) = n = O(n) = \Omega(n) = \Theta(n)$

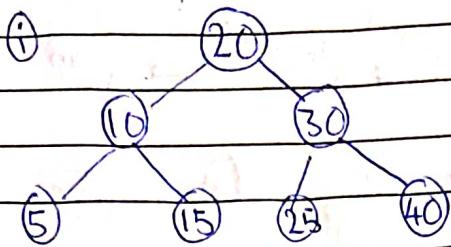
Avg. Case: $A(n) = \frac{n+1}{2} = O(n) = \Omega(n) = \Theta(n)$

Thus, we can look for upper 'O', lower 'Ω' or avg 'Θ' bound for ANY function irrespective of CASE type.



Date: _____

2. Binary Search Tree:



Here, Searching is done inside the tree, similar to simple Binary Search.

$$n = \text{no. of nodes} = 7$$

$$\text{Height of tree} \approx \log n \approx \log 7 = 3$$

$\hookrightarrow \text{In}(i)$

Now, Best Case: Search ends at root node : $B(n) = 1$

Worst Case: Search for leaf node : $W(n) = \text{Height}$
of tree = $\log n$

Avg. Case: (Not discussed)

* Divide and Conquer Strategy:

A strategy is applied to solve a computational problem based on the type of problem and suitability.

Various other strategies:

- i) Dynamic Programming
- ii) Backtracking
- iii) Branch & Bound
- iv) Greedy Technique

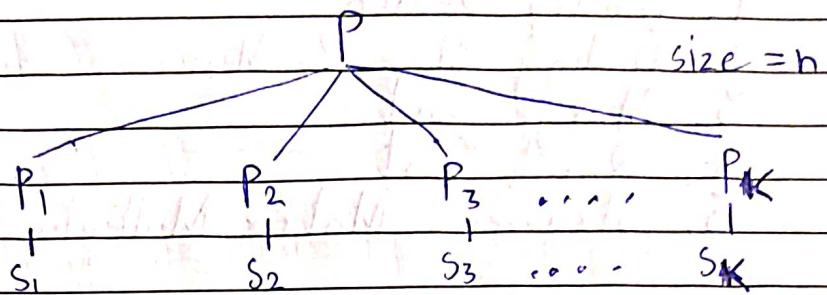
There is no fixed formula to apply strategies but there are guidelines to follow.

Now, we discuss the 'Divide and Conquer' Strategy.



Date: _____

Given a problem 'P' of size 'n', given that the problem is large,



we divide problem into sub-problems, solve it recursively (individually) to get solutions and then combine these solutions to obtain solution for the large problem.

If a sub-problem is also large we apply divide & conquer on it too to get sub-sub-problems and the process continues till a small easy problem is obtained.

The 'sub-problem' however, should be the same type as of that of the original 'large problem'. Thus, these problems are solvable 'recursively'.

• General Algo:

DAC(P) {

 if (small(P)) {

 S(P);

 }

 else {

 divide P into $P_1, P_2, P_3, \dots, P_k$

 Apply DAC(P_1), DAC(P_2), ... //Recursion

 Combine (DAC(P_1), DAC(P_2), ...)

 }

}



Date: _____

Examples of problems solvable by Divide & Conquer approach:

1. Binary Search
2. Finding Max & Min
3. Merge Sort
4. Quick Sort
5. Strassen's Matrix Multiplication

• Recursion:

Divide & Conquer is an 'algorithmic paradigm'.

Recursion is a 'programming paradigm' that is used to solve problems using the 'divide & conquer' strategy.

The 'divide & conquer' approach results in a 'recursive' algorithm which is then implemented using recursion.

Thus, in recursion a function calls itself by passing a 'decreasing' or 'dividing' or any other such 'changing' parameter. This continues until the 'base case' that is smallest solvable sub-problem is reached.

The solutions of each call are combined in the preceding calls until eventually the required solution is returned by the first call.

• Recurrence Relations:

Represents recursion logic for the purpose of understanding and analyzing complexity of recursive algorithms.

We will now see types of recurrence relations and the different methods to solve those.



Date: _____

Methods to solve Recurrence Relations:

- i) Substitution / Iterative / Plug & Chug Method
- ii) Recursive Tree Method
- iii) Master's Theorem
- iv) Akra Bazzi

Types of Recurrence Relations:

Commonly Solved:

- i) Linear Recurrence Relations
- ii) Divide Recurrence Relations

Other Types:

- Nonlinear, Homogenous, Non-homogenous,
- First-order, Higher-order, Full history.

• Examples of Tracing & Solving Recurrence Relations:

(i) void Test(int n) { ... T(n) } * Recursive Tree:

if ($n > 0$) {

printf("%d", n); }

Test(n-1); ... T(n-1)

}

Test(3) - 1

3

Test(2) - 1

3

Test(1) - 1

1

Test(0)

∴ From Recursive Tree,

we see Test(3) executes 3+1 calls

∴ For Test(n) we have $(n+1)$ calls

∴ $O(n)$ is the Time Complexity



Date: _____

* Tracing Recurrence Relation from algo and solving:

$$\therefore T(n) = T(n-1) + 1 \dots \text{from algo}$$

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

* Substitution / Iterative / Plug & Chug method:

We know for $n > 0$,

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

We also know, parameter is decrementing by 1.

$$\therefore T(n-1) = T(n-2) + 1 \dots \text{(ii)}$$

$$\text{And } \therefore T(n-2) = T(n-3) + 1 \dots \text{(iii)}$$

$$\text{Substitute (iii) in (ii) } \therefore T(n-1) = T(n-3) + 2 \dots \text{(iv)}$$

$$\text{Substitute (iv) in (i) } \therefore T(n) = T(n-3) + 3$$

Now we see the trend, if we continue 'k' times

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

$$\text{Assume } 'n-k=0' \therefore T(n) = T(n-n) + n$$

$$\therefore n=k \qquad \qquad \qquad = T(0) + n$$

$$= 1 + n$$

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

∴ Time complexity is $O(n)$.



Date: _____

```

ii) void Test(int n) {
    if (n > 0) {
        for (i = 0; i < n; i++) {
            printf("%d", i);
        }
        Test(n - 1);
    }
}

```

∴ Tracing:

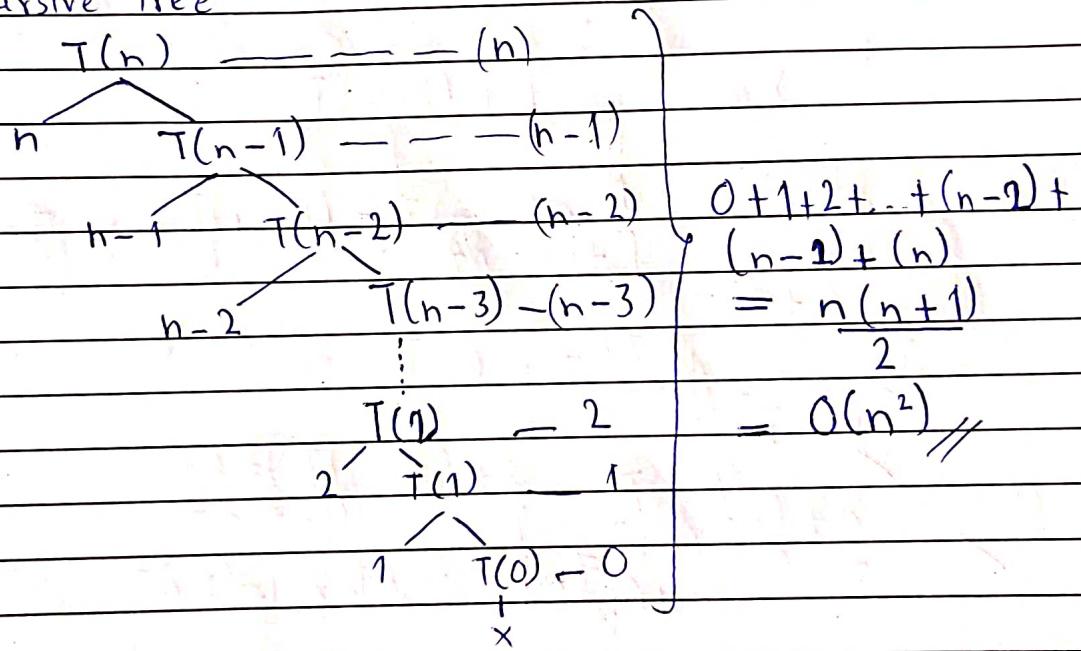
$$T(n) = T(n-1) + \underbrace{1}_{n+1} + \underbrace{2}_{n}$$

Using Asymp. notation,

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

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

∴ Recursive Tree



\therefore Using Subst. on Relation.

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

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

$$T(n-2) = T(n-3) + n-2 \quad \dots \text{iii}$$

$$\text{Subst in (ii)} : T(n-1) = T(n-3) + n-2 + n-1 = T(n-3) + 2n-3$$

$$\text{Subst. in } \textcircled{1} : T(n) = T(n-3) + 2n - 3 + n = T(n-3) + 3n - 3$$

\therefore Trend is: $T(n) = T(n-k) + kn - k$ Possibly Inaccurate

Another way: \rightarrow (Preferred)

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

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

i Assume $n=k$ and it runs 'k' times

$$T(n) = T(n-k) + (n-(k-1)) + \dots + (n-1) + n$$

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

$$= 1 + 1 + 2 + \dots + n$$

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

(iii) void Test(int n) { $\rightarrow T(n)$

if ($n > 0$) { $\rightarrow 1$

for ($i=1$; $i < n$; $i=i*2$) { $\rightarrow (\log n) + 1$

printf("%d", i); $\rightarrow \log n$

}

Test($n-1$); $\rightarrow T(n-1)$

}

}

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

$$\therefore T(n) = T(n-1) + \log n \dots \text{(i)}$$

$$T(n-1) = T(n-2) + \log(n-1) \dots \text{(ii)}$$

$$T(n-2) = T(n-3) + \log(n-2) \dots \text{(iii)}$$

$$\text{Subst. in (ii)} : T(n-1) = T(n-3) + \log(n-2) + \log(n-1)$$

$$\text{Subst. in (ii)} : T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$$

\therefore Trend if it runs 'k' times:

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



Date: _____

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

$$\begin{aligned}
 \therefore T(n) &= T(n-n) + \log(n-(n-1)) + \log(n-(n-2)) + \dots + \log(n-1) + \log n \\
 &= 1 + \log 1 + \log 2 + \dots + \log n \\
 &= 1 + \log(1 * 2 * 3 * \dots * (n-1) * n) \\
 &= 1 + \log(n * (n-1) * \dots * 2 * 1) = 1 + \log(n!)
 \end{aligned}$$

★: We don't have a tight bound (Θ) for ' $n!$ '

However, we know that upper-bound (O) that is $O(n^n)$ will hold true for ' n '.

As we consider

$$\therefore T(n) = 1 + \log(n^n) = 1 + n \log n \\ = O(n \log n)$$

iv) Algo Test(int n) { T(n) }

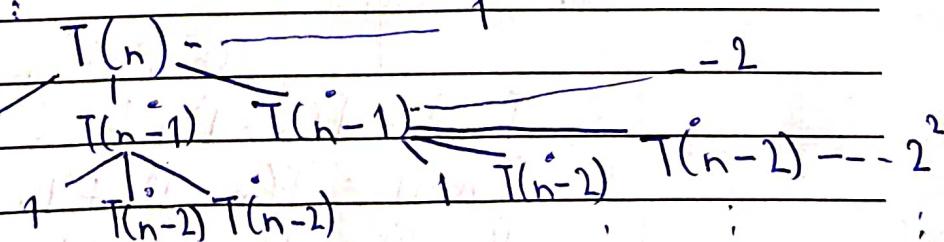
if ($n > 0$) {
 print f("%d", n); }
 $T(n) = 2 T(n-1) + 1$

Test($n-1$); — — T($n-1$)

Test_{n-1}; T(n-1)

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

i) Recursive Tree:



Geometric Progression

Series

$$T(0) \quad T(0) \quad T(0) \quad T(0)$$

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



• Basics of Geometric Progression: (Summation)

If,

$$a + ar + ar^2 + ar^3 + \dots + ar^k = \frac{a(r^{k+1} - 1)}{r - 1}$$

$$\therefore \text{Eg. } 1 + (1)2 + (1)2^2 + (1)2^3 + \dots + (1)2^k \\ = (1)(2^{k+1} - 1)$$

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

↓
Sum of 'k+1' terms in GP.

Now,

Assume ' $n-k=0$ ' i.e., $n=k$

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

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

$$= O(2^n) \rightarrow \text{Exponential}$$

∴ Subst. method

$$\text{We know, } T(n) = T(n-1) + 1 \dots (i)$$

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

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

$$\text{Subst. in (ii) } \therefore T(n-1) = 2[2T(n-3) + 1] + 1$$

$$\text{Subst. in (i) } \therefore T(n) = 2[2[2T(n-3) + 1] + 1] + 1$$

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

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

Let the loop run 'k' times

$$\therefore T(n) = 2^k T(n-k) + 2^{k-1} + \dots + 2^1 + 2^0$$

Assume, ' $n-k=0$ '

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

$$= 1 + 2 + 2^2 + \dots + 2^n$$

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

$$= 2 \cdot 2^n - 1 = O(2^n)$$

∴ Time Complexity is $O(2^n)$.



Date: _____

- Master theorem for Decreasing (Linear) Functions:

From previous examples, we know patterns:

$$\begin{aligned} T(n) &= T(n-1) + 1 \quad O(n) \\ T(n) &= T(n-1) + n \quad O(n^2) \quad \text{Multipled by } n \\ T(n) &= T(n-1) + \log n \quad O(n \log n) \\ T(n) &= 2T(n-1) + 1 \quad O(2^n) \\ T(n) &= 3T(n-1) + 1 \quad O(3^n) \quad \text{Multipled by exponents.} \\ T(n) &= 2T(n-1) + n \quad O(2^n \cdot n) \end{aligned}$$

∴ Generalizing:

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

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

If $a=1$, we get complexity: $O(n^{k+1})$ or $O(n*f(n))$

If $a > 1$, we get complexity: $O(n^k \cdot a^{\frac{n}{b}})$ or $O(f(n) * a^{\frac{n}{b}})$
eg: $T(n) = 2T(n-3) + \log n \rightarrow O(\log n * 2^{\frac{n}{3}})$
 $\& T(n) = nT(n-1) + c \rightarrow O(n^n)$

If $a < 1$, we get complexity: $O(n^k)$ or $O(f(n))$

- Brief info of parameters & functions in recursion:
For a parameter 'n'

If call in recurrence relation is:

- $(n-1) \rightarrow$ Decreasing (Linear) func.
- $(n/2) \rightarrow$ Dividing function
- $(\sqrt{n}) \rightarrow$ Root function (Non-linear)



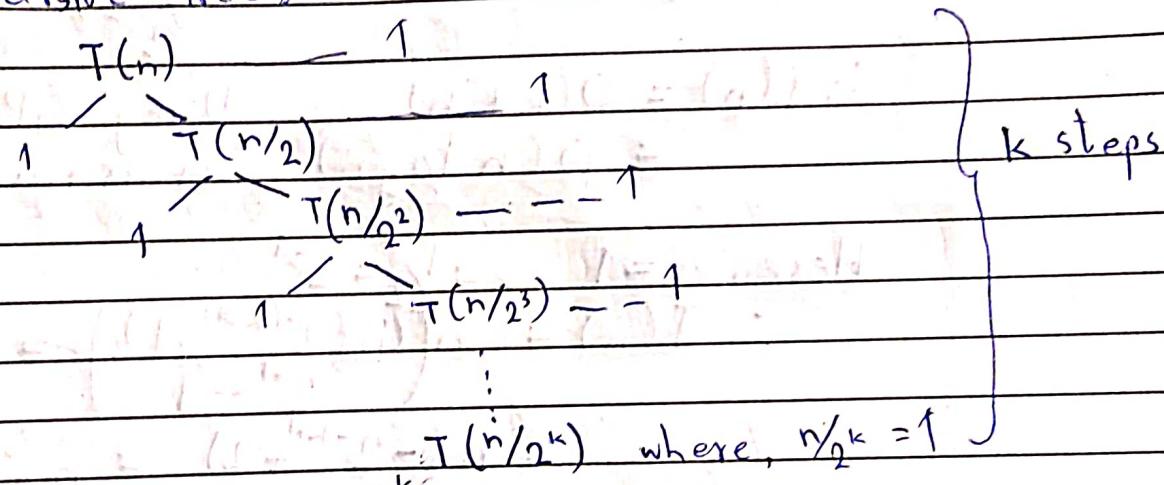
Date: _____

• More Examples (Dividing) Functions:

⑤ Algo Test (int n) {
 if ($n > 1$) {
 printf ("%d", n);
 Test ($n/2$);
 }
}

$$\therefore \text{Relation: } T_n = \begin{cases} 1 & n=1 \\ T(n/2) + 1 & n>1 \end{cases}$$

∴ Recursive Tree:



$$\because n/2^k = 1, \quad n = 2^k \quad \therefore \log n = k \log 2 \quad \therefore k = \log_2 n$$

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

∴ Subst. method:

$$T(n) = T(n/2) + 1 \dots (i)$$

$$T(n/2) = T(n/2^2) + 1 \dots (ii)$$

$$T(n/4) = T(n/2^3) + 1 \dots (iii)$$

$$\text{Subst. in (ii), } T(n/2) = T(n/2^3) + 1 + 1$$

$$\text{Subst. in (i), } T(n) = T(n/2^3) + 1 + 1 + 1 = T(n/2^3) + 3$$

$$\therefore \text{Trend for 'k' times: } T(n) = T(n/2^k) + k$$

$$\begin{aligned} \text{Assume 'n} - 2^k = 0, \quad n = 2^k, \quad \therefore T(n) &= T(1) + \log n \\ \text{and } k &= \log_2 n \end{aligned}$$

$$= 1 + \log n = O(\log n)$$



Date: _____

• More Basics of Geometric Progression:

If we have a sequence,

$$a + ar + ar^2 + ar^3 + \dots + ar^{n-1}$$

n-terms

Then, $a = \text{first term}$, $r = \text{common constant} = \frac{t_{n+1}}{t_n}$

$$t_n = n^{\text{th}} \text{ term} = ar^{n-1}$$

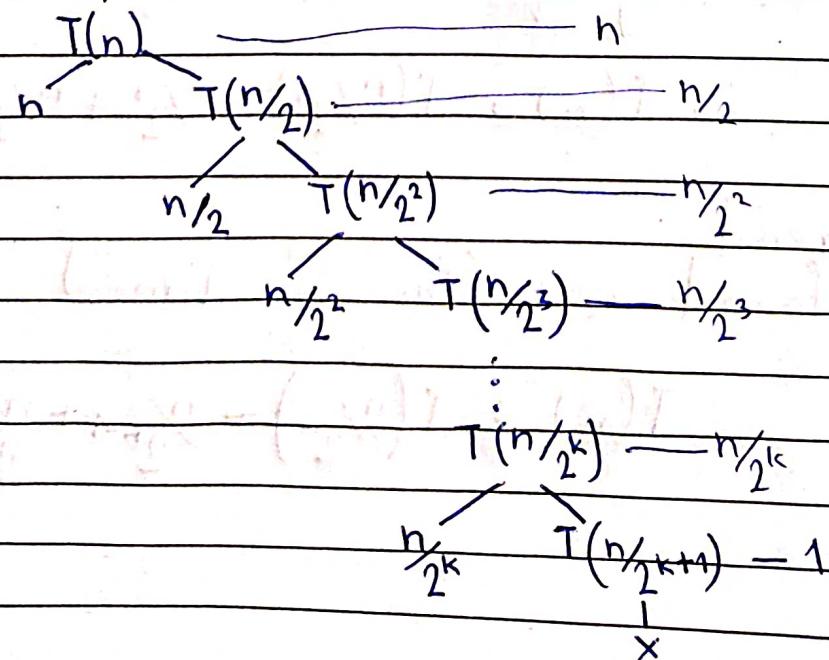
$$S_n = \text{Sum of 'n' terms} = \frac{a(r^n - 1)}{r - 1} = \frac{a(1 - r^n)}{1 - r}$$

$$= na$$

when $r=1$ \downarrow useful when $r > 1$ \downarrow useful when $r < 1$

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

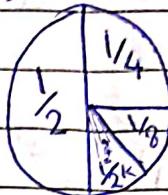
Recursive Tree:





Date: _____

$$\begin{aligned}\therefore T(n) &= 1 + n + \frac{n}{2} + \frac{n}{2^2} + \dots + \frac{n}{2^k} \\&= 1 + n \left(1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^k} \right) \\&= 1 + n \left(1 + \sum_{i=1}^k \frac{1}{2^i} \right) \\&= 1 + n(1+1) \\&= 2n+1 \\&= O(n)\end{aligned}$$



$$\Rightarrow 1$$

Subst. Method:

$$\begin{aligned}T(n) &= T(n/2) + n \quad \text{(i)} \\T(n/2) &= T(n/2^2) + n/2 \quad \text{(ii)} \\T(n/4) &= T(n/2^3) + n/2^2 \quad \text{(iii)}\end{aligned}$$

Substituting eq (iii) in (ii),

$$\therefore T(n/2) = T(n/2^3) + n/2^2 + n/2 \dots \text{(iv)}$$

Substituting eq(iv) in (i),

$$\therefore T(n) = T(n/2^3) + n/2^2 + n/2 + n \dots$$

∴ Trend

found as : (for 'k' times)

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



Date: _____

Assume $n/2 = 1$,

$$\therefore T(n) = T(1) + (2 + 4 + 8 + \dots + n)$$

$$\begin{aligned} T(n) &= \cancel{T(1)} + \frac{1}{2} (2^k - 1) = \cancel{T(1)} + \frac{2^k - 1}{2} \\ &= \cancel{T(1)} + (2^k - 1) \\ &= O(n) \end{aligned}$$

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

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

$$\begin{aligned} &= 1 + 2n \\ &= O(n) \end{aligned}$$

(vii) void Test(int n) {

if (n > 1) {

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

stmt;

}

Test(n/2);

Test(n/2);

}

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

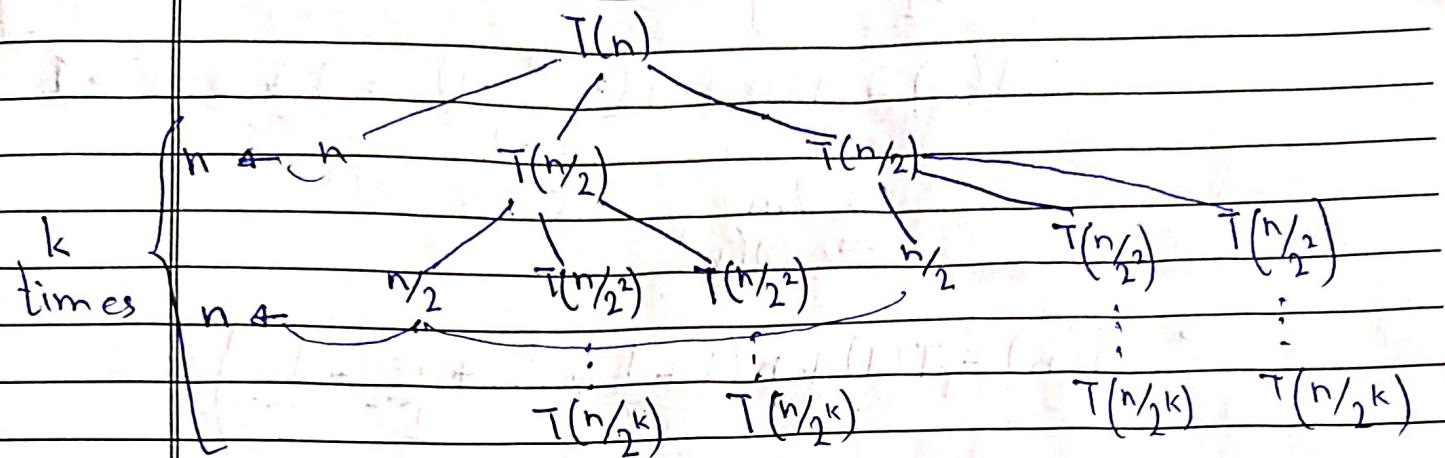
$$= 2T\left(\frac{n}{2}\right) + n$$

$$\therefore \text{Relation: } T(n) = \begin{cases} 1 & n=1 \\ 2T\left(\frac{n}{2}\right) + n & n>1 \end{cases}$$



Date: _____

Recursive Tree:



$$\therefore T(n) = kn \quad \text{and assuming } n/2^k = 1, k = \log_2 n$$

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

Subst. Method:

$$T(n) = 2T(n/2) + n \quad \text{(i)}$$

$$T(n/2) = 2T(n/2^2) + n/2 \quad \text{(ii)}$$

$$T(n/4) = 2T(n/2^3) + n/2^2 \quad \text{(iii)}$$

$$\begin{aligned} \text{Subst. in (ii), } T(n/2) &= 2[2T(n/2^3) + n/2^2] + n/2 \\ &= 2^2 T(n/2^3) + n/2 + n/2 \end{aligned}$$

$$\begin{aligned} \text{Subst. in (i), } T(n) &= 2[2^2 T(n/2^3) + n] + n \\ &= 2^3 T(n/2^3) + 3n \end{aligned}$$

$$\therefore \text{Trend for 'k' steps: } T(n) = 2^k T(n/2^k) + kn$$

$$\text{Assume, } \frac{n}{2^k} = 1, \therefore k = \log_2 n$$

$$\begin{aligned} \therefore T(n) &= n T(1) + n \log n \\ &= n + n \log n \end{aligned}$$

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



Date: _____

• Master's Theorem for Dividing Functions:

General Form: $T(n) = aT(n/b) + f(n)$

where,

$$a \geq 1 \text{ and } f(n) = O(n^k \log^p n)$$

$$b > 1$$

The two important elements: i) $\log_b a$, ii) k

Case 1: $\log_b a > k$, then $O(n^{\log_b a})$

Case 2: $\log_b a = k$, then

- i) If $p > -1$ then, $O(n^k \log^{p+1} n)$
- ii) If $p = -1$ then, $O(n^k \log(\log n))$
- iii) If $p < -1$ then, $O(n^k)$

Case 3: $\log_b a < k$ then,

- i) If $p \geq 0$ then, $O(n^k \log^p n)$
- ii) If $p < 0$ then $O(n^k)$

Eg:

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

Here, $a = 2$, $b = 2$, $f(n) = O(1) = n^0 \log^0 n$
 $\therefore k = 0$, $\log_b a = \log_2 2 = 1$

$\therefore \log_b a > k \dots$ Case 1,

Then, Time complexity is $O(n^1)$

ii) $T(n) = 4T(n/2) + n$

Here, $a = 4$, $b = 2$, $f(n) = O(n^1)$, $k = 1$, $\log_b a = 2$

$\therefore \log_b a > k \dots$ Case 1

Then, Time complexity is $O(n^2)$



Date: _____

$$(iii) T(n) = 8T(n/2) + n$$

 $a = 8, b = 2, \log_b a = 3, f(n) = O(n), k = 1$ $\therefore \log_b a > k \dots (\text{Case 1, } T(n) = O(n^3))$

$$(iv) T(n) = 9T(n/3) + 1$$

 $a = 9, b = 3, \log_b a = 2, f(n) = O(1), k = 0$ $\therefore \log_b a > k \dots (\text{Case 1, } T(n) = O(n^2))$

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

 $\log_b a = 1, k = 1$ $\therefore \log_b a = k \dots (\text{Case 2})$

Here, $f(n) = O(n^1 \log^0 n)$, $p = 0 > -1$

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

$$(vi) T(n) = 8T(n/2) + n^3$$

 $\log_b a = 3, k = 3, p = 0 > -1$ $\therefore \log_b a = k \dots (\text{Case 2, } T(n) = O(n^3 \log n))$

$$(vii) T(n) = 2T(n/2) + n/\log n$$

 $\log_b a = 1, k = 1, p = -1 = -1$ $\therefore \log_b a = k \dots (\text{Case 2, } T(n) = O(n \log(\log n)))$

$$(viii) T(n) = 2T(n/2) + n/\log^2 n$$

 $\log_b a = 1, k = 1, p = -2 < -1$ $\therefore \log_b a = k \dots (\text{Case 2, } T(n) = O(n))$

$$(ix) T(n) = T(n/2) + n^2, \log_b a = 0, k = 2, p = 0$$

 $\therefore \log_b a < k \dots (\text{Case 3, } T(n) = O(n^2))$



Date: _____

• More Examples (Root) Functions:

Viii) $T(n)$ — void Test(int n) {
 if ($n > 2$) {
 $T(\sqrt{n})$ — statement;
 }
}

\therefore Relation: $T(n) = \begin{cases} 1 & n=2 \\ T(\sqrt{n}) + 1 & n > 2 \end{cases}$

$$T(n) = T(\sqrt{n}) + 1 = T(n^{1/2}) + 1 \dots (i)$$

$$T(\sqrt{n}) = T(n^{1/2^2}) + 1 \dots (ii)$$

$$T(\sqrt{\sqrt{n}}) = T(n^{1/2^3}) + 1 \dots (iii)$$

$$\text{Subst. in } (ii), \quad T(\sqrt{n}) = T(n^{1/2^3}) + 1 + 1$$

$$\begin{aligned} \text{Subst. in } (i), \quad T(n) &= T(n^{1/2^3}) + 1 + 1 + 1 \\ &= T(n^{1/2^3}) + 3 \end{aligned}$$

\therefore Trend for 'k' times:

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

$$\text{Assume, } n^{1/2^k} = 2^m, \quad m = \log_2 n$$

~~$$\frac{m}{2^k} \log 2 = \log 2$$~~

$$\therefore T(n) = T(2^{m/2^k}) + k$$

$$\text{Assume, } 2^{m/2^k} = 2$$

$$\therefore \frac{m}{2^k} \log 2 = \log 2 \quad \therefore m = 2^k$$

$$\therefore k \log 2 = \log m, \quad k = \log_2 m$$

$$\begin{aligned} \therefore T(n) &= T(2) + \log_2 m \\ &= 1 + \log(\log n) \\ &= O(\log(\log n)) \end{aligned}$$