

Analytic of Algorithms

Module 1 - Introduction to Algorithms

* Algorithm as a Technology

- An algorithm is simply a set of steps used to complete a specific task.
- If computer memory would be infinitely fast and free, studying different solutions to a problem wouldn't be a necessity.
- But memory is cheap and not free so finding different solutions of a problem and choosing the cost-efficient method is a must.

* Algorithm Design:

- The important aspects of algorithm design include:
 - ① creating an efficient algorithm.
 - ② solving the problem using efficient time and space.
- Both time and space cannot be minimized simultaneously, if one is more other is less.

* Problem Development

→ some common steps toward problem development

- ① Analyzing the Problem definition.
- ② Development of a model.
- ③ Specification of Algorithm
- ④ Designing an Algorithm
- ⑤ Checking correctness of the algorithm.
- ⑥ Analysis of the Algorithm.
- ⑦ Implementation of the Algorithm.
- ⑧ Program Testing.

* Characteristics of Algorithms

- ① Algorithms must have a unique name
- ② Algorithms should have explicitly defined set of inputs and outputs
- ③ Algorithms are well-ordered unambiguous operations.
- ④ Algorithms halt after a finite amount of time.

* Pseudocode.

→ A pseudocode gives high-level description of an algorithm without plain text and without need of actual syntax.

* Difference in Algorithm, Pseudocode and Program.

- ① Start from the leftmost element of arr[] and one by one compare each element of arr[]
- ② If n matches the index, return the index.
- ③ If n doesn't match with any of the elements return -1

Q3

→ Pseudocode.

function LinearSearch(list , searchItem)

for index from 0 → length(list)

If list[index] == SearchItem

return index

Else

pass;

Return -1.

End function.

→ Program

```
int LinearSearch (int arr[], int index):  
    for (int i = 0; i < size; i++)  
        if (arr[i] == arr[index])  
            return i;  
    else  
        pass;  
    return -1;
```

* Rate of Growth of Functions -

- The rate at which running time increases as a function of input is called Rate of Growth.
- Example - Total cost = cost of car + cost of candy

∴ Total cost = cost of car
(By approximation)

* Common Running times

④	Time complexity	Name	Example
①	1	Constant	Adding element in LL
②	$\log n$	Logarithmic	Finding element in a sorted array
③	n	Linear	Linear search
④	$n \log n$	Linear Log.	Quick sort / Merge sort
⑤	n^2	Quadratic	Shortest path b/w two nodes
⑥	n^3	Cubic	Matrix Multiplication
⑦	2^n	Exponential	Towers of Hanoi

* Asymptotic Notations and Running Time

→ The order of growth of running time is called Asymptotic Running Time.

Some Notations are:-

I] The Big-O Notation

$f(n) = O(g(n))$ iff there are two positive constants c and n_0 such that

$$|f(n)| \leq c|g(n)| \text{ for all } n > n_0$$

If $f(n)$ is positive

$$0 \leq f(n) \leq cg(n) \text{ for all } n > n_0$$

→ As n increases, $f(n)$ grows no faster than $g(n)$ so $g(n)$ is an asymptotic upper bound of $f(n)$.

For example

$$n^2 + n = O(n^3)$$

$$\therefore f(n) = n^2 + n ; g(n) = n^3$$

We know $\forall n > 1 \quad n^2 \leq n^3$

If $a \leq b$ then $n^a \leq n^b$

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

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

2] The Big-Omega Notation (Ω)

$f(n) = \Omega(g(n))$ iff there are two positive constants c and n_0 such that:

$$|f(n)| \geq c|g(n)| \text{ for } n > n_0$$

If $f(n)$ is positive

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

→ As n increases, $|f(n)|$ grows no slower than $|g(n)|$. In other words, $g(n)$ is an asymptotic lower bound on $f(n)$.

Example $n^3 + 4n^2 = \Omega(n^2)$

$$f(n) = n^3 + 4n^2 \quad g(n) = n^2$$

$$\text{we know } n^3 \leq n^3 + 4n^2$$

$$\text{and for } n \geq 1, n^2 \leq n^3.$$

$$\text{so } n^2 + n^3 \leq n^3 + 4n^2 \\ \therefore n^2 \leq n^3 + 4n^2.$$

③ The Big-Theta function (Θ)

$f(n) = \Theta(g(n))$ iff there are three constants c_1, c_2, n_0 such that:

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

when $f(n)$ is +ve

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

→ When n increases $f(n)$ grows at the same rate as $g(n)$ so $g(n)$ is an asymptotic tight bound on $f(n)$.

Example $n^2 + 5n + 7 = \Theta(n^2)$

when $n \geq 1$

$$n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2$$

$n \geq 0$

$$n^2 \leq n^2 + 5n + 7$$

$n \geq 1$

$$\therefore n^2 \leq n^2 + 5n + 7 \leq 13n^2$$

* Recurrence Relations

→ A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. There are 8 methods to solve these relations:-

- ① Substitution Method.
- ② Recursion-Tree Method.
- ③ Master Method.

* Substitution Method.

It involves the following steps:

- ① Guess the Solution - first, making an educated guess about the form of the solution to the recurrence relation.
- ② Prove the guess:- By using mathematical induction we prove our guess to be true.
- ③ Solve for constants.

Example:

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ n*T(n-1) & \text{if } n>1 \end{cases}$$

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

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

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

$$\therefore T(n) = n(n-1)(n-2)(n-3) \dots 3 \times 2 \times T(1)$$

$$T(n) \therefore n(n-1)(n-2) \dots 3 \times 2 \times 1$$

$$T(n) = n * n\left(1 - \frac{1}{n}\right) * n\left(1 - \frac{2}{n}\right) \dots n\left(\frac{3}{n}\right)$$

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

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

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

Example 2

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T\left(\frac{n}{2}\right) + n & \text{if } n \geq 1 \end{cases}$$

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

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \quad \text{--- (2)}$$

∴ From ① and ②

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

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

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

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

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

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

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

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

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

$$\therefore \log n = K$$

$$\therefore T(n) = n \cdot 1 + n \log n$$

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

Example 3 - $T(n) = T(n-1) + \log n \dots n \geq 1$

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

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

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

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

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

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

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

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

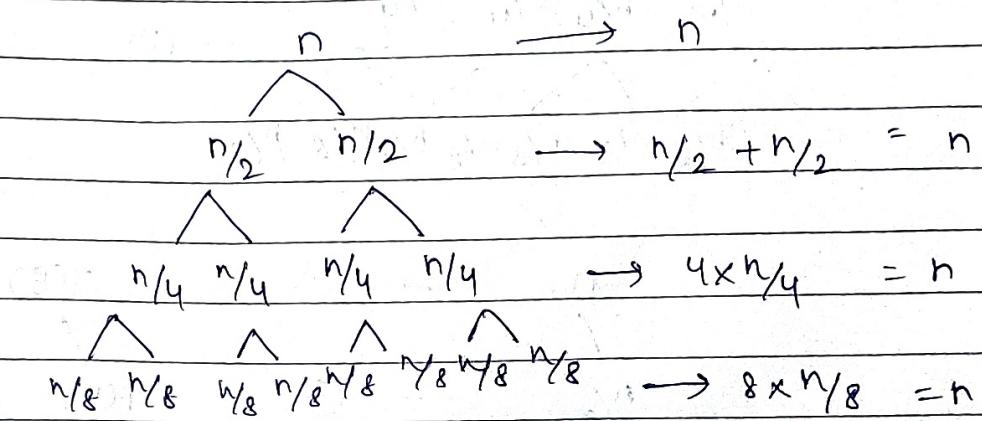
$$\therefore O(n \log n)$$

② Recursion Tree method.

- This method involves drawing a tree to represent recursive calls made by the algorithm. Each node in a tree represents a single call in the algorithm.

→ To determine time complexity, we calculate the amount of work done at each level of the tree and then sum all the work done.

Example $\rightarrow T(n) = 2T(n/2) + b$



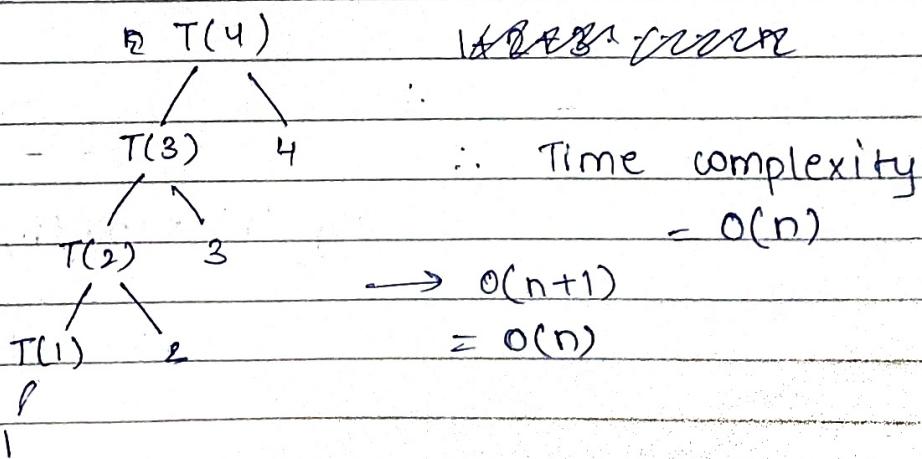
\therefore Work done \times height of tree

\therefore Total work $= n \times \log n$.

$\therefore O(n \log n)$

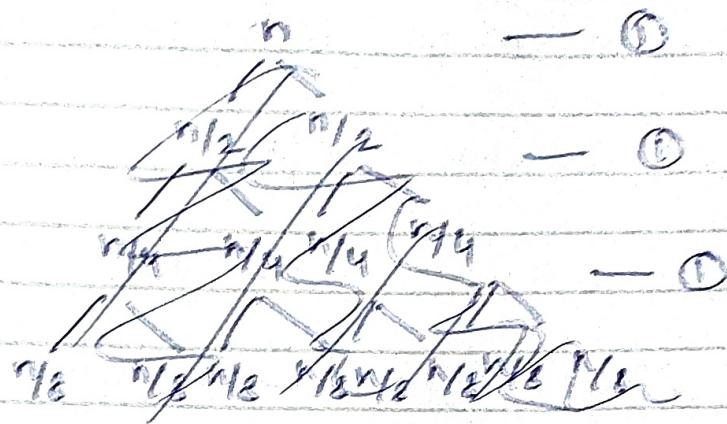
Another Example

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



classmate
Date _____
Page _____

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



$$n \rightarrow ① \quad T(8)$$

$$n/2 \rightarrow ① \quad T(4)$$

$$\begin{array}{c} n/4 \\ | \\ n/8 \end{array} \rightarrow ① \quad \begin{array}{c} T(2) \\ | \\ T(1) \end{array}$$

∴ Height of tree is $\log_2 n$

∴ Number of nodes = $\log_2(n) + 1$

$$\text{Total operations} = 1 + \log_2\left(\frac{n}{2}\right)^1 + \log_2(4)$$

$$+ \log_2(8)$$

$$= \log_2(n) + \log_2(n/2) + \log_2(n/4) \dots$$

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

* Master Method.

→ It can solve various problems of the form

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

where $a \geq 1$ and $b > 1$

Solutions

$$T(n) = \Theta(n^{\log_b a} [u(n)])$$

$u(n)$ depends on $f(n)$

$$R(n) = f(n)$$

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

If $R(n)$

$$n^R \rightarrow R > 0 \text{ then } O(n^R)$$

$$n^R \rightarrow R < 0 \text{ then } O(1)$$

$$n^R \rightarrow R = 0 \text{ then } (\log_2 n)^{i+1}$$