Analysis of Algorithms

Module 1 - Introduction to Algorithms

* Algorithm as a Technology

→ An algorithm is simply a set of steps used to complete a specific tasks.

→ If computer memory would be infinitely fast and free, studying different solutions to a problem wouldnt 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 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 x matches the index, return the index.

③ If x doesnt match with any of the elements return -1

④

→ 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

	Time complexity	Name	Example
①	1	Constant	Adding element in LL
②	logn	Logarithmic	finding element in a sorted array
③	n	Linear	Linear search
④	nlogn	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:-

1] The Big-O Notation

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

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

If $f(n)$ is positive

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

→ 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 $\forall \; n \geq 1 \quad n^2 \leq n^3$

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:

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

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

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

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

And for $n \geq 1 \quad n^2 \leq n^3$.

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 consta
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 3 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 ~~index~~ induction we prove our guess to be true.

③ Solve for constants

Example:

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

$T(n) = n * T(n-1)$ — (1)

$T(n-1) = (n-1)\,T(n-2)$ — (2)

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

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

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

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

$n\left(\frac{2}{n}\right) \times \frac{1}{n}$

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

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

Example 2

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

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

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

∴ From ① and ②

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

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

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

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

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

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

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

$$\log n = \log 2^K$$

$$\log n = K \log_2 2$$

$$\therefore \log n = K$$

$$\therefore n \cdot 1 + n\log n$$

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

Example 3 - $T(n) = T(n-1) + \log n$. $n > 1$

$n = 1$.

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

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

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

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

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

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

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

$\therefore O(n\log n)$

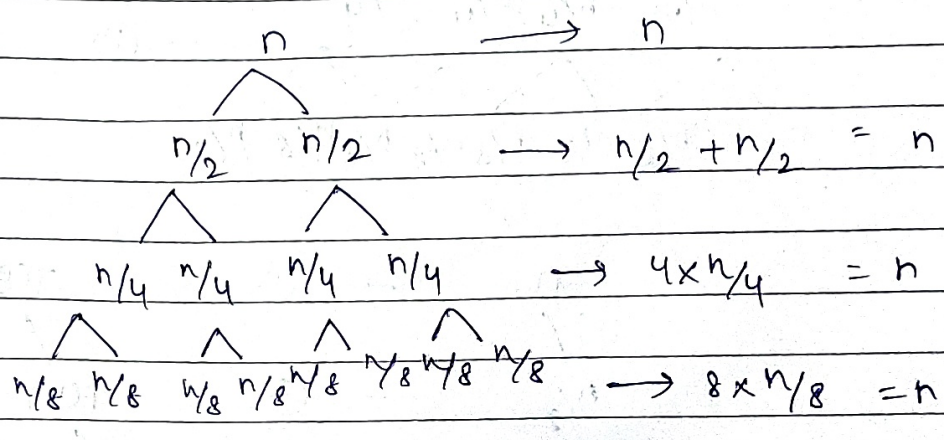
---

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 → $T(n) = 2T(n/2) + n$



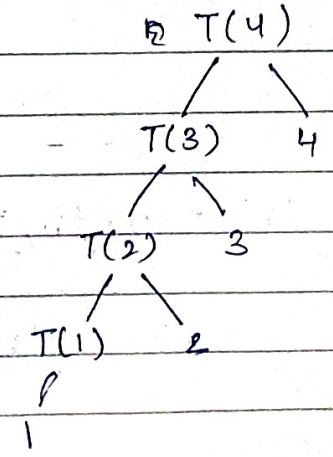
∴ n work done x height of tree

∴ Total work = $n \times \log n$.

∴ $O(n \log n)$

Another Example

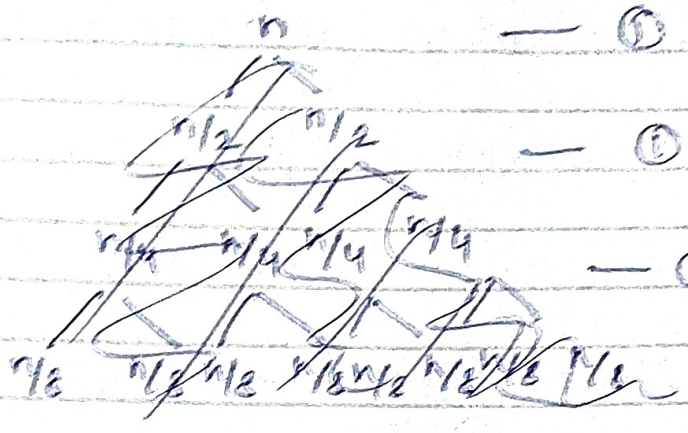
$T(n) = T(n-1) + n$.

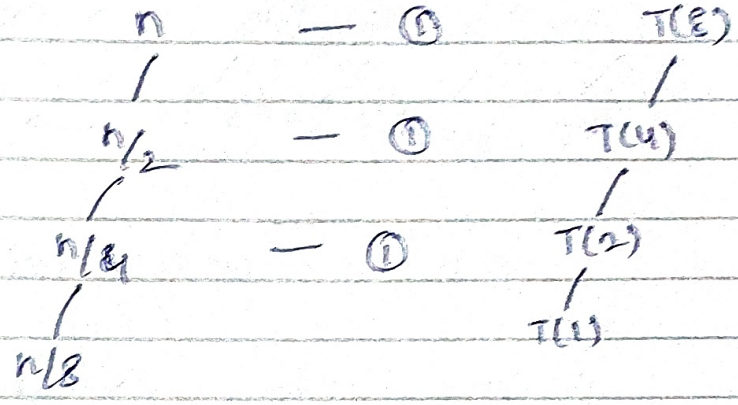


∴ Time complexity
$= O(n)$

→ $O(n+1)$
$= O(n)$

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





∴ Height of tree is $\log_2 n$

∴ Numbers of nodes = $\log_2(n) + 1$

Total operations = $1 + \log_2\left(\frac{2^i}{2}\right)^1 + \log_2(4) + \log_2(8)$

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

∴ $T(n) = \Theta(\log n)$

* Master Method.

→ It can solve various problems of the form

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

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

Solutions

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

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

$$R(n) = \frac{f(n)}{n^{\log_b a}}$$

if $R(n)$

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

$nR \rightarrow R < 0$ then $O(1)$

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

Page No.
Date

* Calculating Time complexity of any Algorithm

① Loops. → Example.

```
for (int i=0 ; i<=n; i++) // Running n times
{
    x = y+2; { Suppose this step takes
}                'c' time }
```

$\therefore T_n = c * n$

$\therefore \boxed{T_n = O(n)}$

② Nested Loop.

```
for (int i=0; i<n ; i++)  // Running n times
  { for(int j=0; j<n; j++) // Running n times.
    {
        x = y+2; { Takes 'c' time }
    }

}
```

$\therefore T_n = c * n * n$

$T_n = c * n^2$

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

(3) Sequential Statement

i) a = a+b → O(1) time {constant}

ii)
```
for(int i=0 ; i<=n ; i++)
{
    x = y+2;
}
```

(4) If - else statements.

```
if (condition)
{
    ..... O(n)
}
else                                  → O(n²)
{
    ..... O(n²)
}
```

Priority Order:

According to time taken

Q. O(1) < O(logn) < O(n) < O(nlogn) < O(n²) < O(n³) < O(eⁿ) < O(n!)

Algorithm.	Best	Worst	Avg.
① Selection Sort	$\Omega(n^2)$	$O(n^2)$	$\Theta(n^2)$
② Bubble Sort	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$
③ Insertion Sort	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$
④ Heap Sort	$\Omega(n \log n)$	$O(n \log n)$	$\Theta(n \log n)$
⑤ Quick Sort	$\Omega(n \log n)$	$O(n \log n)$	$\Theta(n^2)$
⑥ Merge Sort	$\Omega(n \log n)$	$O(n \log n)$	$\Theta(n \log n)$
⑦ Bucket sort	$\Omega(n+k)$	$O(n+k)$	$\Theta(n^2)$
⑧ Radix sort	$\Omega(nk)$	$O(nk)$	$\Theta(nk)$