ASSIGNMENT - 01

Que 1. What is ADA ? What is need to study Algorithm ? Explain in detail.

Ans ⇒ "ADA" is a high level, general purpose programming language developed for safety-critical and embedded systems.

ADA refers to different things depending on the content one common interpretation is the programming language.

It is named after Ada Lovelace who is considered the world's first computer programmer.

NEED TO STUDY ALGORITHM. —

(1) Problem Solving :-

Algorithms provides systematic and structured approaches to problem solving

(2) Efficiency :-

Efficient algorithms are essential for optimizing resource usage such as time & space.

(3) Programming :-
In programming we need to chosen or design an algorithm to implement a particular functionality.

(4) Interviews & Coding challenges :-
Many technical interviews for software engineering position involves algorithmic problem solving.

(5) Optimization :-
In various application such as database management, network routing & graphics processing.

(6) Research & innovation :-
Advancement in computer science & technology often rely on the development of new and more efficient algorithms.

(7) Real-World Application :-
Algorithms are everywhere in our daily lives, from search engines & social media algorithms to recommendation system & navigation apps.

Q2. Give the divide and conquer solution for quicksort and analyze its complexity?

Quicksort is a well known sorting algorithms that follows the divide and conquer paradigm.

Here's a high level overview of the divide and conquer solution for quicksort.

(i) Divide :-
Choose a "pivot" element from the array partition the array into two subarrays

(ii) Conquer :-
Recursively apply the quick sort algorithm to the subarrays created in the previous step.

(iii) Combine :-
- As the subarrays are sorted in place, no explicit combining is needed.
- The entire array is sorted when the recursive calls are complete.

```
Partition (A, p, r)
{
    x = A[r]
    i = p-1
    for (j = p to r-1)
{
    if (A[j] ≤ x).
    {
        i = i+1
        exchange
            A[i] ↔ A[j]
    }
}
    exchange
    A[i+1] ↔ A[r]
    return i+1
}
// Algorithm
        quicksort (A, p, r)
        {
        if (p<r)
            q = partition (A, p, r)
            quicksort (A, p, q-1)
            quicksort (A, q+1, r)
```

Master Theorem provides solution for three cases :-

Case 1 :-

if $f(n)$ is $O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.

where $\log_b a$ is the logarithmic term in the recurrence relation.
then $T(n)$ is $\Theta(n^{\log_b a})$

Case 2 :-

if $f(n)$ is $\Theta(n^{\log_b a} \cdot \log^k n)$ for some constant $k \geq 0$ where $\log_b a$ is the logarithmic term in recurrence relation then
$T(n)$ is $\Theta(n^{\log_b a} \cdot \log^{k+1} n)$

Case 3 :-

$f(n)$ is $\Omega(n^{\log_b a + \epsilon})$ for some constant $k < 1$ & sufficiently large n & if $a f(n/b)$ is asymptotically bounded by a polynomial in n
then

$T(n)$ is $\Theta(f(n))$

Asymptotic Notations.

→ By using asymptotic notations, we can calculate time complexity of an algo.

→ By using asymptotic notations, we can calculate Best case, Avg case & worst case time complexity of an algo.

⇒ There are 5 types of asymptotic notations.

1) Big oh (O)
2) Big omega (Ω)
3) Theta notation (Θ)
4) Little oh (o)
5) Little omega (ω)

1] Big Oh (O) :-

→ $O(n)$ is the formal way to express the upper bound of an algo's running time.

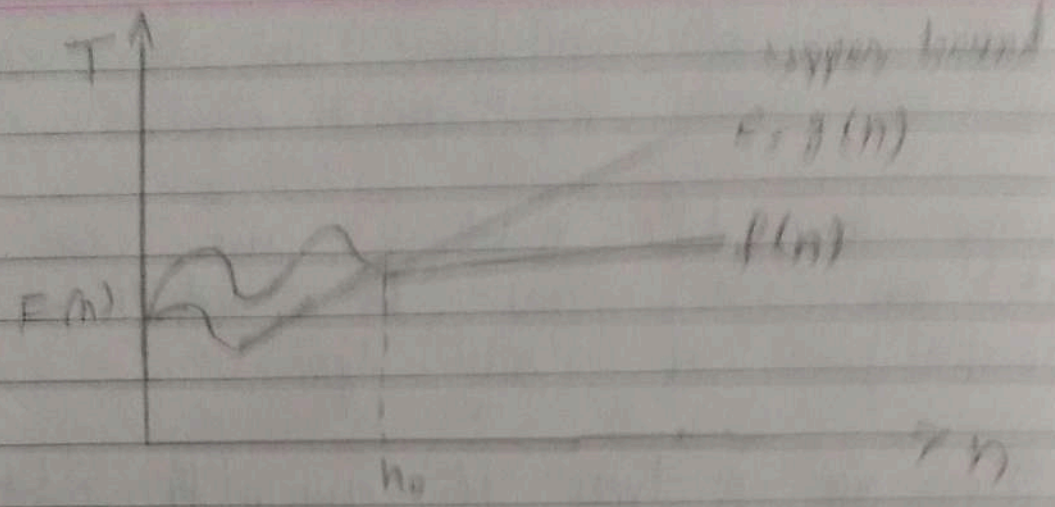
→ It measures the worst case time complexity or, longest amount of time an algo can possibly take to complete.

⇒ $f(n) \leq c \cdot g(n)$

$n \geq n_0$.

$c > 0, \; n_0 \geq 1$

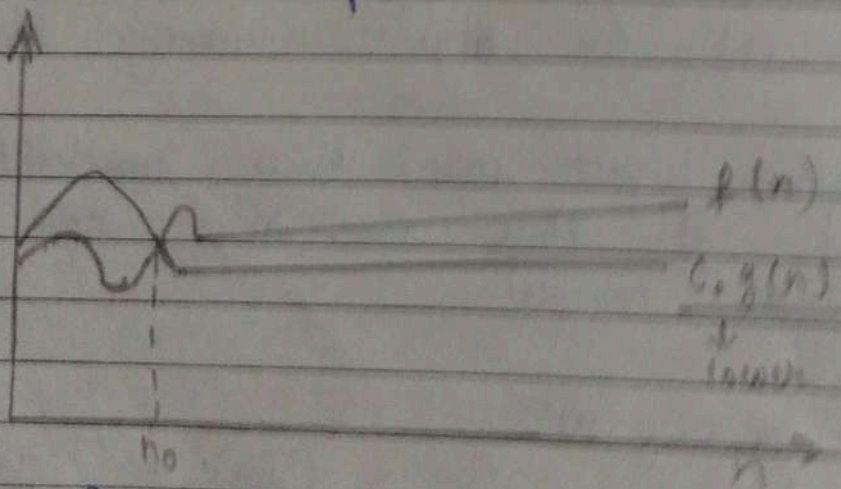
$\Rightarrow f(n) = O(g(n))$



2] Big Omega (Ω).

→ The $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time.

→ It measure the best case time complexity or best possible amount of time an algo can take to complete.



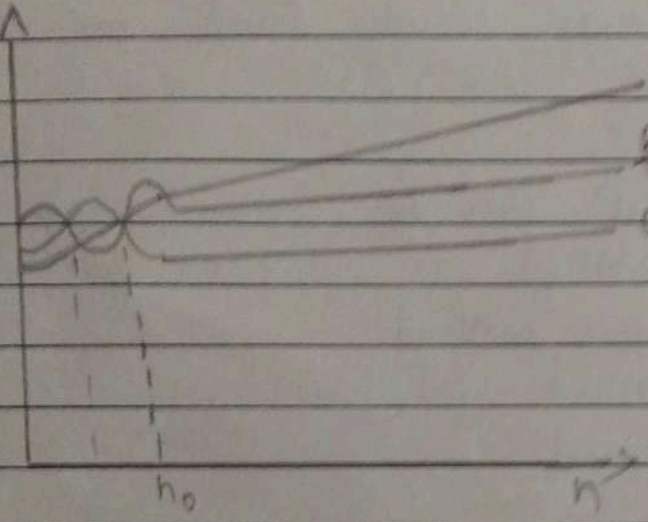
$f(n) \geq c \cdot g(n)$

$n \geq n_0$ → $f(n) = \Omega(g(n))$

$c > 0, \; n_0 \geq 1$

quicksort can be

② Big Theta (Θ) Notation :-
The $\Theta(n)$ is the formal way to express both the lower bound & upper bound of an algo's running time.



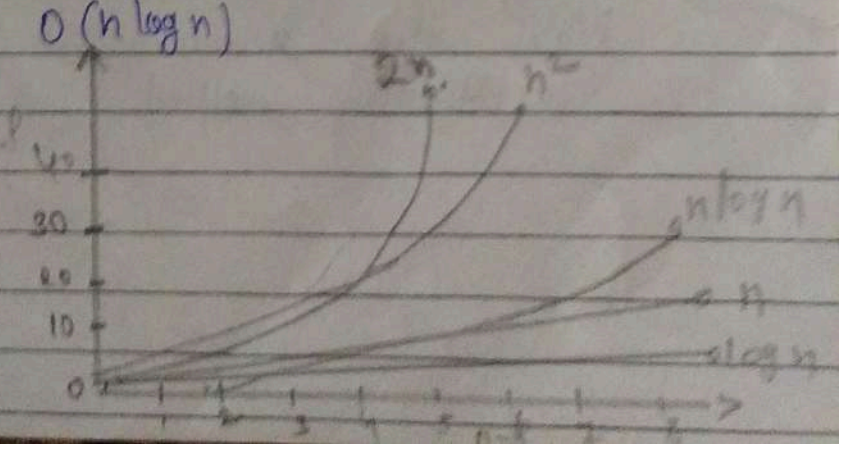
→ This graph represents the time growth of an algo. w.r.t. input size 'n'.

$$c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$$
$$n \geq n_0$$
$$c_1, c_2 > 0 \qquad n_0 \geq 1$$

$$\Rightarrow \boxed{f(n) = \Theta(g(n))}$$

It is used in avg case

Complexity :-
- $O(1)$: constant
- $O(n)$: Linear
- $O(n^2)$: quadratic
- $O(n^3)$: cubic
- $O(2^n)$: exponential
- $O(\log n)$



→ If an algorithm has a space complexity of $n(n)$, it means that at least in the best case the space requires grows linearly with input size.

→ If an algorithm has a time complexity upper and lower bound of the running time are proportional to $n \log n$. Providing a precise characteristics of its efficiency.

Que 4 Write all the three cases of Master Theorem for the equation

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

The master theorem is a mathematical tool used to analyze the time complexity of recursive algorithms that follow a specific form. The general form of the recurrence relation is

$$T(n) = a \cdot T(n/b) + f(n)$$

where

$T(n)$ = time complexity of algorithm.

a = no. of subproblems in each recursive call

b = cost of dividing problem & combining results

Date:
P. No:

Que 5 How can we prove that Strassen's matrix multiplication is advantageous over ordinary matrix multiplication.

Ans Strassen's algorithm for matrix multiplication is known for its efficiency in certain cases offering advantages over the standard algorithm in terms of time complexity.

(1) Theoretical Analysis - Time complexity

Comparison - Compare the time complexity of Strassen's algorithm. ($O(n^{2.8})$) with the standard matrix multiplication algorithm ($O(n^3)$)

(2) Practical Experiments :-

Implement both Algorithms :-
Write implementation of both the standard matrix multiplication algorithm and strassens algorithms.

Input Size Variation :-
Conduct experiment with matrices of varying size.

Execution time measurement :-
Measure and compare the execution time of both algorithms for different matrix size.

Graphical representation :-
Created graphs or charts to visually represent the execution times for different matrix size.

Practical Consideration

Memory usage -
Strassen's algo involves additional memory usage due to the recursive decomposition of matrix.

13/3/24