Assignment-1

Date: / / Page no:

Q.1 What is ADA? What is the need to study Algorithms? Explain detail.

Ans:- Analysis and Design of Algorithms (ADA):- ADA is a branch of computer science that focuses on the study of Algorithms, their design, analysis of their efficiency, and their applications in solving computational problems. An algorithm is a sequence of steps to solve a problem. It acts like a set of instructions on how a program should be executed. Thus, there is no fixed structure of an algorithm. Design and Analysis of Algorithms covers the concepts of designing an algorithm as to solve various problems in computer science and information technology, and also analyse the complexity of these algorithms designed.

Need to study Algorithm.

1. Problem solving:-

Algorithms provide systematic approaches to problem-solving. They help break down complex problems into smaller, manageable steps.

2. Efficiency:-

Understanding algorithms helps in developing efficient solutions.

Date: / / Page no:

3. Optimization :-
Algorithm are crucial for optimizing the performance of software.

4. Correctness :-
Algorithms play a key role in ensuring the correctness of software.

5. Foundation for Programming :-
Algorithms form the foundation of programming.

6. Data structure :-
Algorithms are closely tied to data structures understanding algorithms helps in choosing appropriate data structure to represent and organize data efficiently.

Q.2 Give the Divide and Conquer solution for Quicksort and ~~analysis~~ analyze its complexity.

Ans :- Divide & Conquer is a top-down approach to Solve a problem. the algorithm which involves Divide & Conquer technique involves 3 steps.

- Divide the original problem into a set of sub problems.
- Conquer every sub problem individually recursive

Date: / / Page no:

- Combine the solutions of thise subproblems to get the solution of original problems.

Quicksort is a popular Sorting algorithms that ~~ext~~ employs the divide - and - conquer the basic idea is to divide the array into smaller subarrays, recursively sort the subarray, and then Combine the main steps includs :-

i) Divide
- Choose a pivot element from the array.
- Partition the array into two subarrays. elements less than the pivot and elements greater than the pivot.

ii) Conquer :-
Recursively apply quicksort to the subarrays.

iii) Combine.
No explicit combining steps is needed since the array is sorted in place.

Eg :-

1	2	3	4	5
1	5	0	6	(4)

i=0 j=1 r=5

r=5
x is the pivot.
A[r] = x
so x = 4

Date: / / Page no:

→ // Partition function

```
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)
}
```

Date: / / Page no:

Complexity Analysis

① Time complexity

- In avg. & best cases, time complexity is $O(n \log n)$. where n is no. of elements in array.
- In worst case time complexity is $O(n^2)$

② Space Complexity :-

- The space complexity is $O(\log n)$ in avg. case due to recursive calls.
- In worst case, it can become $O(n)$ when the recursive calls form a skewed tree.

Q.3 Define asymptotic notations. Give different notations used to represent the complexity of algorithms.

Ans ⇒ Asymptotic notations :-

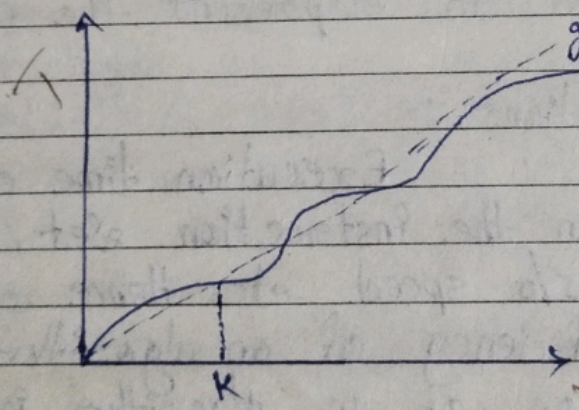
Execution time of an algorithm depends on the instruction set, processor speed, disk I/o speed etc. Hence, we estimate the efficiency of an algorithm Asymptotically. time function of an algorithm is represented by $T(n)$, where n is the input size. Different types of asymptotic notations are used to represent the complexity of an algorithm. Following asymptotic notations are used to calculate the running time complexity of an algorithm.

Date: / / Page no:

- O - Big on Notation
- O - Big Omega Notation
- O - Big theta Notation.

- Big on. O :- Asymptotic Upper bound the notation O(n) is the formal way to express the upper bound of an algorithm's running time. It is the most commonly used notation. A function f(n) can the represented is the order of g(n) that is O(g(n)), if there exists a value of possitive integer n as no & a positive Constant c such that.

$f(n) \leq c\,g(n)$ for $n \geq n_0$ in all case ~~g(n)~~

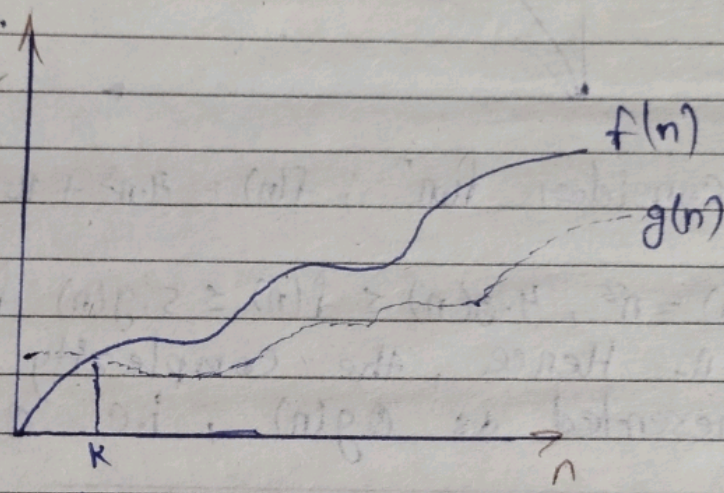


Hence, function g(n) is an upper bound for function f(n), as g(n) is faster then f(n)

- Big O Omega Ω : Asymptotic lower bound the notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm, running time

Date: / / Page no:

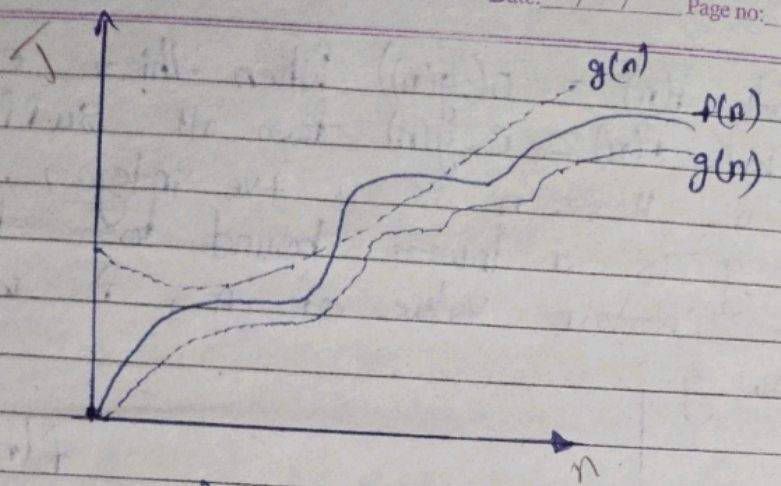
We say that $f(n) = \Omega(g(n))$ when there exists constant that $f(n) \geq c \cdot g(n)$ for all sufficiently large value of n. Here n is a +ve integer, It means function g is a lower bound for function f after a certain value of n, f will never go below g.



Ex let Consider functions, $f(n) = 4.n^3 + 10.n^2 + 3n + 1$
Considering $g(n) = n^3$, $f(n) \geq 4 \cdot g(n)$ for all the values of $n > 0$.
Hence, the complexity of $f(n)$ can be represented as $\Omega(g(n))$, i.e. $\Omega(n^3)$

- Theta, Θ :- Asymptotic tight Bound the notation $\Theta(n)$ is the formal way to notation $O(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running times.
We say that $f(n) = \Theta(g(n))$ when there exists Constants c_1 & c_2 that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all sufficiently large value of n. Here n is a the integer this means function g is a tight bound for function f.

Date: / / Page no:



Ex: Let consider fun^n ; $f(n) = 4.n^3 + 10.n^2 + 5n + 1$

Consider $g(n) = n^3$, $4.g(n) \leq f(n) \leq 5.g(n)$ for all the large value of n. Hence, the complexity of $f(n)$ can be represented as $\Theta g(n)$, i.e. $\Theta(n^3)$.

Q.4 ~~Ans~~ Write all the three cases of Master theorem for the equation:

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

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

Where, $a >= 1$ and $b > 1$,

n = Size of the problem

a = number of sub problem in the recursion

$\frac{n}{b}$ = size of the sub problems based on the assumption that all sub problems are of the same size.

$f(n)$ = represents the cost of work done outside

the recursion → $O(n^k \log^p n)$, where $k >= 0$ and p is a real number: if the recurrence relation is in the above given form, then there are three cases in the master theorem to determine the asymptotic notations.

(i) If $a > b^k$, then
$T(n) = O(n^{\log_b a})$ [$\log_b a = \log a / \log b$]

(ii) If $a = b^k$
(a) If $p > -1$, then
$T(n) = O(n^{\log_b a} \log^{p+1} n)$

(b) If $p = -1$, then
$T(n) = O(n^{\log_b a} \log \log n)$

(c) If $p < -1$, then
$T(n) = O(n^{\log_b a})$

(iii) If $a < b^k$,
(a) If $p >= 0$, then
$T(n) = O(n^k \log^p n)$

(b) If $p < 0$, then
$T(n) = O(n^k)$

Q.5 How can we prove that strassen's Matrix Multiplication is advantageous over ordinary Matrix Multiplication.

Ans => It is used to solve the matain multiplication problem.

- The usual matrix multiplication method multiplies

Date: / / Page no:

each row with each column to achieve the product matrix. Time Complexity ~~the~~ taken by that approach is $O(n^3)$. since it takes three loops to multiply.

- This method was introduced to reduces the time complexity from $O(n^3)$ to $O(n^{\log 7})$

⇒ Iterative method of Matrix Multiplication

Let x is a matrix of $p \times q$

y is a matrix of order $q \times r$

$z = [x]_{p \times q} \quad [y]_{q \times r}$

z has the order $p \times r$

Algorithm :- Matrix multiplication (x, y, z)

```
for i = 1 to p
    for j = j to r
        z[i, j] = 0
        for k = 1 to q to
            z[i, j] = z[i, j] + x[i, k] * y[k, j]
```

Complexity = $O(n^3)$

SMM Algorithm : With this algorithm the time consumption can be improved a little bit.

Note :- It can be applied only on square matrix Where 'n' is a power of 2,

Date: / / Page no:

Order of both the matrices are $n \times n$.

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \& \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

$M_1 : (A + C) \times (E + F)$
$M_2 : (B + D) \times (G + H)$
$M_3 : (A - D) \times (E + H)$
$M_4 : A \times (F - H)$
$M_5 : (C + D) \times E$
$M_6 : (A + B) \times H$
$M_7 : D \times (G - E)$

then

$I : M_2 + M_3 - M_6 - M_7$
$J : M_4 + M_6$
$K := M_5 + M_7$
$L := M_1 - M_3 - M_4 - M_5$

Bhubaneswar
11/3/24.