DATE
PAGE

Name :- Bhumika Kala
class :- CS-A
Subject :- Analysis Design of Algorithms (ADA)
Sub Code :- CS-402

Assignment - I

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

Ans ADA :- Analysis & Design of algorithm is a branch of computer science that focuses on the study of algorithm, their design, analysis of their efficiency and their application 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.

The main aim of designing an algorithm is to provide a optimal solution for a problem.

One computer problem might have several versions of a solution. In this case, every approach taken to solve the computer problem is correct. However choosing the best suited solution will improve the efficiency of the program.

→ Design techniques of an algorithm

1) Greedy Approach.

2) Divide & conquer Approach
3) Dynamic Programming Approach
4) Randomization Approach
5) Approximation Approach
6) Recursive Approach
7) Branch and Bound Approach.

→ Characteristics of an Algorithm.

1) Unambiguous :- Algorithm should be clear & unambiguous.

2) Input :- An algorithm should have 0 or more well-defined inputs.

3) Output :- An algorithm should have 1 or more well-defined output and should match the desired output.

3) Finiteness :- Algorithms must terminate after a finite no of steps

4) Feasibility :- Should be feasible with the available resources

5) Independent :- An algorithm should have step-by-step direction which should be independent of any programming code.

Que:-2 Give the Divide and Conquer Solution for QuickSort and analyze its complexity.

Ans Quick Sort is a popular sorting algorithm that uses the divide and conquer approach.

It works by selecting a point pivot element and partitioning

the array into two sub-arrays, one with elements smaller than the pivot other. This process is repeated recursively until the array is sorted.

The complexity of quick sort depends on the choice of the pivot element. In the average case, the time complexity is $O(n \log n)$, where n, is the number of elements in the array.

However, in the worst case, when the pivot is consistently chosen as the smallest or largest element, the time complexity can be $O(n^2)$.

Algorithm of Quick Sort :-

```
QUICKSORT (A, P, r)

1) if P < r
2)   q = PARTITION (A, P, r)
3)   QUICKSORT (A, P, q-1)
4)   QUICKSORT (A, q+1, r)

PARTITION (A, P, r)

1)   x = A[r]
2)   i = P-1
3)   for j = P to r-1
4)     if A[j] ≤ x
5)       i = i+1
6)       exchange A[i] with A[j]
7)   exchange A[i+1] with A[r]
8)   return i+1
```

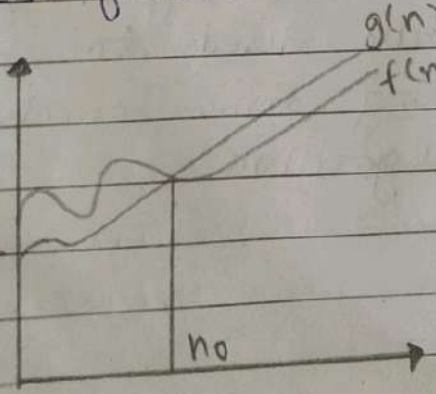
Que:-3) Define Asymptotic Notations. Give different notations used to represent the complexity of algorithms.

Ans 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.

Types of Notations :-

1) Big Oh (O) :- The notation O(n) is the formal way to express the upper bound of an algorithms running time. It is the most commonly used notation. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

Function g(n) is an upper bound for function f(n), as g(n) grows faster than f(n).

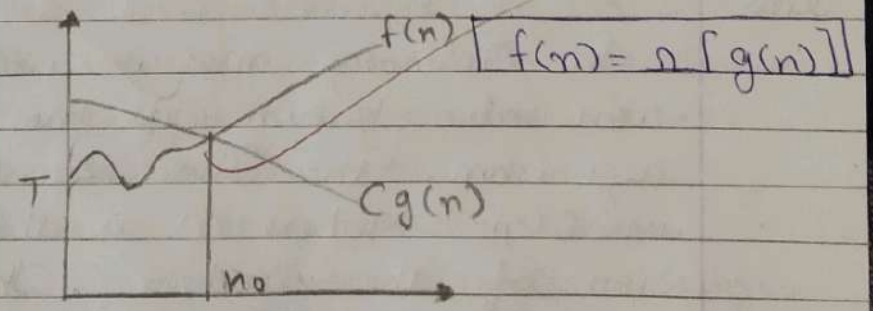


$\because n \geq n_0$ then $f(n) \leq \{g(n)\}$ where C is constant

$$f(n) = O[g(n)]$$

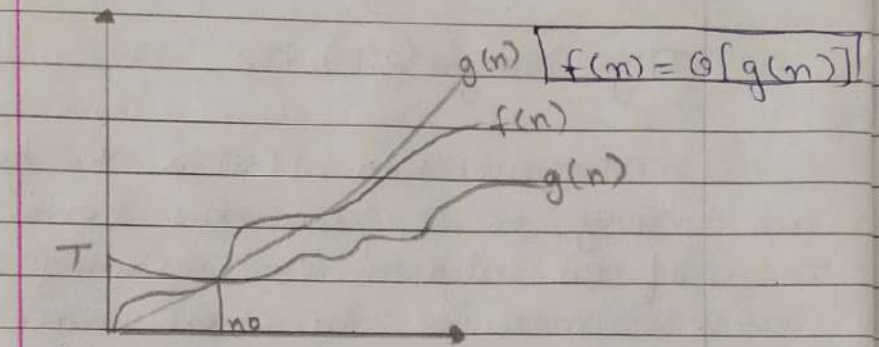
2) Big Omega (Ω) :-

The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete



3) Big Theta (Θ) :-

The notation $\Theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time.



Que:-1 How can we prove that Stressen's Matrix Multiplication is advantageous over ordinary matrix multiplication.

Ans

In ordinary matrix multiplication also known as the naive algorithm, the time complexity is $O(n^3)$ where n is the dimension of the matrix. This means that as the size of the matrix increases, the time required to multiply them increases significantly

On the other hand, Strassen's Matrix multiplication algorithm has a time complexity of $O(n^{\log_2(7)})$, which is approximately $O(n^{2.81})$. This complexity is lower than the ordinary matrix multiplication algorithm for larger matrix size.

Strassen's algorithm achieves this improvement by dividing the matrix into smaller submatrix and performing fewer multiplications. It recursively breaks down the matrix multiplication into smaller subproblems, reducing the number of multiplications required.

formula =?

Ques5) Write all the three cases of Master Theorem for the equation

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

Ans The master theorem is used in calculating the time complexity of recurrence relations (divide and conquer algorithms) in a simple and quick way. 2)

If $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function, then the time complexity of a recursive relation is given by-

1) If $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

If the cost of solving the sub-problems at each level 3)

increases by a certain factor the value of $f(n)$ will become polynomically smaller than $n^{\log_b a}$. Thus, the time complexity is expressed by the cost of the last level i.e $n^{\log_b a}$.

2) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} * \log n)$.

If the cost of solving the sub-problem at each level is nearly equal then the value of $f(n)$ will be $n^{\log_b a}$. Thus value of $f(n)$ the time complexity will be $f(n)$ times the total number of levels i.e. $n^{\log_b a} * \log n$.

3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$, then $T(n) = \Theta(f(n))$

If the cost of solving the subproblems at each level decreases by a certain factor the value of $f(n)$ will become polynomically larger than $n^{\log_b a}$

Thus, the time complexity is expressed by the cost of $f(n)$.

[ADA] lab work.

Iterative

Date 05/03/23

Binary Search :-

```
#include <iostream>
using namespace std;

int binarySearch (int arr[], int left,
    int ~~left~~ right, int target) {

  while ( left <= right ) {
        int mid = left + ( right - left) /2;

      if (arr [mid ] == target)
          return mid;

      if ( arr [mid ] < target)
          left = mid + 1;

    else
          right = mid - 1;
  }

    return -1;
  }

    int main () {
    int arr [ ] = { 2, 4, 6, 8, 10 };
    int n = sizeof (arr) / size of (arr [0]);
    int target = 10;

  int result = binarySearch (arr, 0, n-1,
        target);

    if ( result == -1 )
        Cout << " element not found";
  else
      Cout << " element found at index
          << result ;
```