classmate
Date
Page

Assignment:- 01

Que:1) what is ADA? what is need to study Algorithms? explain in detail.

Ans:- "ADA" could refer to different things depending on the content one common interpretation is the programming language is ADA, named After ADA lovelace who is considered the worlds first computer Programmer.

ADA is a high-level, general purpose programming language developed for Safety - Critical and embedded Systems.

The need to study algorithms in detail:-

1) Problem Solving:- Algorithms provide Systematic and Structured approches to problem Solving.

2) Efficiency:- Efficient Algorithms are essential for optimizing resource usage such as time and Space.

3) Programming:- As a programmer you'll frequently encounter situations where you need to choose or design an algorithms to implement a particular functionality.

4) Interviews and coding challenges:- Many technical interviews for Software engineering

position involve algorithmic problem solving.

(5) computer science foundation:- Algorithms and Data structures from the foundation of computer science. They are fundamental concepts that every computer scientist or programmer should be familiar with.

6) optimization:- In various optimization application such as database Management, network routing and graphics processing optimization is ~~[illegible]~~ crucial.

7) Research and innovation:- Advancement in computer science and technology often rely on the development of new and more efficiency algorithms.

8) Real-world Application:- Algorithms are everywhere in our daily lives, from search engines and social media algorithms to recommendation systems and navigation apps.

Que-2) Give the divide and conquer Solution for quicksort and analyze its complexity?

Ans:- 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, elements less than the pivot and greater than the pivot.

2) conquer:- Recursively apply the quick sort algorithms to the subarrays created in the previous step.

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

4) Complexity Analysis:- The time complexity of quicksort can be analyzed using the recurrence relation.

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

(i)

(1) Best case:- The Best case scenario occurs when the pivot consistently divides the array into two equal halves $T(n) = 2T(n/2) + O(n)$ The solution to this recurrence relation is $O(n \log n)$

(2) Average case:- on average, if pivot divide the array reasonably well, the recurrence relation remain $O(n \log n)$.

3) worst case:- The worst case Scenario occurs when the pivot is always the smallest or largest element leading to unbalanced partitions

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

The solution to this recurrence relation is $O(n^2)$

Que:-3) Define asymptotic notations, give different notations used to represent the complexity of algorithms.

Ans:- Asymptotic notations are mathematical notations used to describe the behaviour of function as their input approaches infinity.

(1) Big O notation (O) :-

$f(n)$ is $O(g(n))$ if there exist positive constants c and n_0 such that $0 \le f(n) \le c\, g(n)$ for all $n \ge n_0$ intuition - Big O notation represents the uper bound or worst case scenario of the algorithm growth rate.

(2) Omega Notations (Ω):-

$f(n)$ is $\Omega(g(n))$ if there exist positive constant c and n_0 such that $0 \le c\, g(n) \le f(n) \le$ for all $n \ge n_0$ Intution- omega notation represent the lower-bound or best case scenario of the algorithm growth rate.

(3) Theta Notations (Θ) :- $f(n)$ is $\Theta(g(n))$ if and only if it is both $O(g(n))$ and $\Omega(g(n))$

~~Intitution~~ Intuition- theta notation provides a tight bound on the growth rate indicating both the upper and lower bound are closely matched.

Example:-

1) if an algorithms has a time complexity of $O(n^2)$ it means that the worst case running time grows ~~grows~~ quadratically with the size of the Input.

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

3) if an algorithms has a time complexity upper and lower bounds of the running time are proportional to $n\log n$ Providing a precise characterization 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 from. The general form of the recurrence relation is given by

$$T(n) = a.T(n/b) + f(n)$$

where:-

→ $T(n)$ is the time complexity of the algorithms.

→ a is the number of subproblems in each recursive call.

→ b is the cost of dividing the problem and combining the result.

→ The Masters theorem provides solution for three cases:-

Case 1:- if $f(n)$ is $O(n^{\log a - e})$ for some constant $f > 0$. 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 $\theta(n^{\log a} \log^k n)$ for some constant $k \geq 0$ where $\log_b a$ is the logarithmic term in the recurrence relation then $T(n)$ is $\theta(n^{\log_b a} \cdot \log^{k+1} n)$

Case 3:- $f(n)$ is $\Omega(n^{\log a + c})$ for some constant $k < 1$ and sufficiently large n and if $a f(n/b)$ is asymptotically bounded by a polynomial in n then $T(n)$ is $\theta(f(n))$.

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 comparision:- compare the time complexi of Strassen's algorithms ($O(n^{2.81})$) with the standard matrix multiplication algorithms ($O(n^3)$)

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 measurment:- measure and compare the Execution time of both algorithms for different matrix size.

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

practical consideration:-

Memory usage:- ~~Streesh~~ Strassen's algorithm involves additional memory usage due to the recursive decomposition of matrices.

11/3/24