7TH FEBRUARY.

ADNAN ANSARI

033

ASSIGNMENT - I - FA

Q.1 Define algorithm and its state of essential characteristics?

→ That term was coined by "persian author".

An algorithm is a set of self-contained sequence of instruction or action that contain finite space or sequence &, that will give us a result to a sperific problem in a finite amount of time.

An algorithm is a effective, efficient and best method which can be used to express solution of any problm within a finite amount of space & time and in a well defined formal languages.

characteristics:

1. Input:
An algorithm must take zero or more inputs that are required for a solution of a prolem.

2. process:
An algorithm must perform certain operations on the input data.

3. Output:
An algorithm should produce certain output after processing the input.

4. ~~fitne~~ finiteness:
An algorithm must terminate after executing certain finite numbes of steps.

5. Effectiveness:
An algorithm should be simple & clear. Each step in the algorithm must be unambiguous, feasible & definite.

Q.2. Explain different types of Algorithm representation?

→ 1. Recursive Algorithm
2. Dynamic Programming algorithm
3. Backtracking Algorithm
4. Divide and conquer Algorithm.
5. Greedy Algorithm
6. Brute force algorithm
7. Randomized algorithm.

1. R.A:- Solve the base case directly and recursive with a simple or easier input every time (A base value is set at the starting for which the algorithm terminates).
It is use to solve the problem which can be broken into simple or smaller problem of same type.
ex:
Factorial using recussion.

2. D.P.A: A dynamic programming Algorithm is also called as dynamic optimization algorithm) remembers the past result & uses them to find new result means it solve complex problems by breaking it down into a collection of simpler subproblem. then solving each of those subproblem only once, & storing their solution for

future use instead of recomputing their solution again.

Example: fibonacci sequence

3. B.A:

- How about we learn backtracking using an ex. let's say we have a problem "Monk" and we divide into four smaller problems "M, O, N, K". It may be the case that the solution of these problems did not get accepted as the solution of "Monk"
- In fact we did not know on which one it depends. So will check each one of them one by one (until) we find the solution of "Monk".
- So basically we attempt solving a subproblem but if we didn't reach the desire sol^n.

ex. Queens problem.

4. Divide & C.A: Divide & conquer consist of a two parts first of all is divide the problem into smaller subproblem of the same type & solve them recursively & then combine them to inform form the solution of the original problem

example: Quick & Merge Sort.

5. Greedy Algorithm:-

Greedy algorithm is an algorithm that solve the problem by taking optimal sol^n at the local level. (without regards for any consequences) with the hope of finding optimal sol^n.

at the global level.
- Greedy Algo. is used to find the optimal solution but it is not necessary that you will definetly find the optimal solution.

Ex. Huffman Tree, Counting Money

6. Brute force Algo.
A brute fore algorithm simply tries all the possibilities until a satisfactory solution is found.
- Such type of algorithm are also used to find the optimal (best) solution as it checks all the possible solutions.

ex. Exact String Matching Algo.

7. Randomized Algo.
A R. Algo uses a random number at least once during the computation to make a decision

ex. Quick Sort.

Q3. Define why analysis of algo is important?

→. solving problems.
- finding the better way to solve a problem. without analyzing an algorithm how do you know whether an algorithm is good or not.

Ex. To sort the numbers, we have many algorithm like insertion sort, selection sort, quick sort, merge sort etc.

- In CS. the analysis of algorithms is the determination of the amount of time, storage &/or other resources necessary to execute them.
- so, the choice of an algorithm is of great imp, which can be made by considering the following factors.
  - programming requirements of an algorithm
  - Time requirement of an algorithm
  - Space requirement of an algorithm.

- Algorithm tend to become shorter, simpler, and more elegant during the analysis process.
- The analysis of an algorithm can help us to understand it better, & can suggest informed improvents.

Q.4. Explain the term in rate of Growth?

→ - Running time analysis is the behaviour of the algorithm in terms of input. How the algorithm is behaving when we keep input increased.
- Rate of growth is noting but the representation of running time & space of an algorithm.

- We want to examine the rate of growth $f(n)$ with respect to standard functions like $\log n$, $n^1, n^2, n^3, 2^n$ etc.

Q.5. Write a short note on asymptotic notation. and what are the guideline for asymptotic notation.

→ Asymptotic notatic are languages that allows us to analyse an algorithm's running time by identifying its behaviours as the input size for the algorithm increase.

- It is used to find out which algorithm is better for the same problem statement.
- following are the Notations.

1. Big-oh (O) notation
2. Big-Omega (Ω) notation
3. Theta (Θ) notation.

1. Big - (oh) notation: Big-O notation gives the tight upper bound of the function. It measore the worst case time complexity or the longest amount of time an algorithm.

$$f(n) <= c \times g(n).$$

2. Big- Omega notation: Big omega notation gives tightes lowes bound of the given function. It measures the best case time complexity or the best amount of time an algorithm possibly take to complete.

$$f(n) >= c \times g(n)$$

Where $c > 0$ & $n_0 >= 1$ the for all $n >= n_0$.

$$f(n) = \Omega(g(n)).$$

3. Theta (θ) Notation: Theta notaties lies between upper bound & lower bound of an algorithm using this we can compute the average amoun of time taken by an algorithm.

This means a function $f(x)$ is theta of function $g(x)$ & these exists three positive constants. C_1, C_2 & n_0 such that

$$0 <= C_1 * g(n) <= f(n) >= C_2 * g(n)$$

for all $n > n_0$.

Q.6. Write a short note on master theorem for subtract and conquer recurrence?

→ Let $T(n)$ be a function defined on positive n as shown below:

$$T(n) \leq \begin{cases} c, & \text{if } n<1 \\ aT(n-b)+f(n), & n>1 \end{cases}$$

for some constants $c, a>0, b>0, k>=0$ and function $f(n)$.

If $f(n)$ is $O(n^k)$, then

1. If $a<1$ then $T(n) = O(n^k)$
2. If $a=1$ then $T(n) = O(n^{k+1})$
3. If $a>1$ then $T(n) = O(n^k a^{n/b})$

Q.7.

Q.7. Write a short note on method of Guessing & Confirming?

→ A method which can be used to solve any recurrence. The basic idea behind this method is,

- ~~In other words, it addresses~~
- let's start by trying to prove an upper bound $T(n) \leq c\, n \log n$:

$$\begin{aligned} T(n) &= \sqrt{n}\, T(\sqrt{n}) + n \\ &\leq \sqrt{n}\, c \sqrt{n} \log \sqrt{n} + n \\ &= n.c \log \sqrt{n} + n \\ &= n.c.1/2.\log n + n \\ &\leq c\, n \log n. \end{aligned}$$

- The last inequality assumes only that $1 \leq c.1/2.\log n$. This is correct if n is sufficiently large & for any constant c. no matter how small. from the above proof, we can see that one guess is correct for upper bound. Now, let us prove the lower bound for this recurrence.

$$\begin{aligned} T(n) &= \sqrt{n}\, T(\sqrt{n}) + n \\ &\geq \sqrt{n}\, k \sqrt{n} \log \sqrt{n} + n \\ &= n.k \log \sqrt{n} + n \\ &= n.k.1/2.\log n + n \\ &\geq k\, n \log n. \end{aligned}$$

- The last inequality assumes only that $1 \geq k\, 1/2.\log n$. This is incorrect if n is sufficiently large & for any constant k.

proving lower bound for $n\sqrt{\log n}$:

$$T(n) = \sqrt{n}\,T(\sqrt{n}) + n$$
$$\geq \sqrt{n}.k.\sqrt{n}\left(\sqrt{\log\sqrt{n}}\right) + n$$
$$= n.k.1/\sqrt{2}\log\sqrt{n} + n$$
$$\neq k\,n\log\sqrt{n}.$$

Q.8 Describe master theorm for divide and conquer method.

→. In divide and conquer approach, the problem is divided into smaller sub-problem & then each prblm is solved independently. the solution of all sub problem is finally merge in order to obtain the solution of an original problem.

The master theorm concerns recurrence relation of the form.

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

where $a \geq 1$ & $b > 1$.

- In the application to the analysis of a recursive algorithm, the constants & function take on the following significance
  - n is the size of the problem
  - a is the no. of subproblem in the recursion
  - n/b is the size of each subproblem.

from above discussion, we understood $\Theta(n \log n)$ is too big. $T(n) = \sqrt{n}\, T(\sqrt{n}) + n \geq n$

Now, let us prove the upper bound for this $\Theta(n)$.

$$\begin{aligned} T(n) &= \sqrt{n}\, T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot c \cdot \sqrt{n} + n \\ &= n \cdot c + n \\ &= n(c+1) \\ &\not\leq cn \end{aligned}$$

from the above induction, we understood $\Theta(n)$ is too small & $\Theta(n \log n)$ is too big

proving upper bound for $n\sqrt{\log n}$:

$$\begin{aligned} T(n) &= \sqrt{n}\, T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot c \cdot \sqrt{n} \left(\sqrt{\log \sqrt{n}}\right) + n \\ &= n \cdot c \cdot 1/\sqrt{2} \log \sqrt{n} + n \\ &\leq c\, n \log \sqrt{n} \end{aligned}$$

proving lower bound for $n\sqrt{\log n}$

$$\begin{aligned} T(n) &= \sqrt{n}\, T(\sqrt{n}) + n \\ &\geq \sqrt{n} \cdot c \cdot \sqrt{n} \left(\sqrt{\log \sqrt{n}}\right) + n \\ &= n \cdot c \cdot 1/\sqrt{2} \log \sqrt{n} + n \\ &\geq c\, n \log \sqrt{n} \end{aligned}$$

- If the recurrence is of the form,

$T(n) = a^k T(n/b) + \theta(n^k \log^p n)$,

where $a >= 1$, $b > 1$, $k >= 0$, & p is a real no.

1. If $a > b^k$ then $T(n) = \theta(n^{\log_b a})$.
2. If $a = b^k$
   a) If $p > -1$ then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
   b) If $p = -1$ then $T(n) = \theta(n^{\log_b a} \log \log n)$
   c) If $p < -1$ then $T(n) = \theta(n^{\log_b a})$
3. If $a < b^k$
   a. If $p >= 0$ then $T(n) = \theta(n^k \log^p n)$
   b. If $p < 0$ then $T(n) = O(n^k)$.

Q.9. briefly explain Master theorem?

→ The Master theorm is used in calculating the time complexity of a recurrence relation of (divide & conque algorithm) in a simple and Quick way.

The Master theorm provides a solution to recurrence relation of the form.

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

- for constant $a > 1$ and $a < 1$ with asymptotically +ve, such recurrence occur frequently in the runtime anlysis of many commonly encountered algorithm.

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^{\log_b a})$

Case 2: If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \log n)$

Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ (& $af(n/b) \leq cf(n)$ for some $c > 1$ for all sufficiently large) then $T(n) = \theta(f(n))$.

Q.10 Explain Runtime algorithm & also explain different cases?

→ • One of the most important aspect of an algorithm is how fast it is. It is the process of determining how process time increases as the size of the problem increases.

• It is not very easy to calculate the exact time requirement for any algorithm as it depends upon various factors like Machine on which algorithm is to be executed, algorithm itself & Input size of the algorithm.

• Because the processor speed in different Machine may be different, so we maintain Concentsate to estimate the execution time of an algorithm irrespective to the processor/Machine

* different Cases:-

To analyze the given algorithm we need to know which input the algorithm takes less time & with which input the algorithm takes a long time.

1. Worst Case:

In the running time of the algorithm is longest for all the inputs then it is called worst case. In this type, the key operations are executed maximum no. of items

2. Best Case:

In the running time of the Algorith is shortest for all the inputs then it is called as best case. In this type, the key operations are executed minimum no. of items.

3. Average Case:

In the running time of an algorithm falls between the worst and best case then it is called as average case. To calculate the average case complexity of an algorithm, we have to take some assumption.