①

16/10/21

Advanced Algorithms

Basics of Analysis: In General time taken by an algorithm grows with size of the input. Thus the need for terms such as "running time" & "size of input" (more carefully).

Algorithm - set of steps

Problem — one problem many sol^n Algorithms.

Time: least amt. of time considered best.

i/p $\xrightarrow{Algo}$ o/p

Ex: Search (array, x) (Linear Search).

Checks for presence of x in array.

Case 1 :→ x is the 1st Element, Search successful. no. of Checks will be 1. (Thus this will be the Best Case).

Case 2 :→ x is not present in array (Search unsuccessful). (Worst Case). (Always consider Worst Case for better results).

No. of Checks = n.
→ Depends on size of i/p.
→ Worst ↓ → Best ↓

Worst Case Time ⇒ Counting no. of Operations.

Rules: Primitive Operations
→ Assignment
→ Calculations
→ Indexing
→ Calling & Return (from a function)

} all take unit time.

Ex:1) $i=3;$ 1 assignment operation 1 unit of time

2) $A[i]=3$; 2 " " 1st unit Indexing, 2nd unit assigning.

Ex: → Sum of digits from 1 till N.

```
Sum_till_N(N)
{
    sum=0;                  ———— 1 unit time, 1 assignment.
    for (i=1; i<=N; i++) → 3 parts. (No. of times = N times).
    {
        sum = sum + i;  (-2N one for addition, one for assigning.
    }
    return sum;  — 1 primitive Operation.
}
```

for: $\underbrace{i=1;}_{1}$ b4 $\underbrace{i<=N;}_{N\ times}$ after Execution $i++$ — $1+(N+1)$ times, N (increments) └ access in reduce the failure. └ Condition fails.

Total Time = $1+1+(N+1)+N+2N+1 = 4N+4$ Total time taken in Worst Case. Time is f^n of input size.

$\therefore T(N) = 4N+4.$

Substitution Method :- Always gives precise results, can be used to solve all recurrences. (not by Master Method). Disadv is involves lot of mathematical Equations.

$$\text{Ex: } T(n) = \begin{cases} 1 & \text{if } n=1 \\ \underbrace{n \cdot T(n-1)}_{\text{Recurrence Equation}} & \text{if } n>1 \end{cases}$$

— base Condition, termination Condition

$T(n) = n * T(n-1)$ — 1 it will decrease in all function calls.

$T(n-1) = (n-1) * T((n-1)-1) = (n-1) * T(n-2)$ — 2

$\left(T(n-2) = (\underbrace{(n-2)-1}_{n-3}) * T(\cancel{(n-2)-2})\right)$ $T(n-2) = (n-2) * T(n-3)$ — 3

Generalized form, Back substitution → $T(n) = n * (n-1) * \underbrace{T((n-1)-1)}_{T(n-2)}$.

→ Mathematical Analysis of Recursive Algorithms

Ex 1. Compute the factorial fⁿ $F(n) = n!$ for an arbitrary non-ve integer n. Since

$$n! = 1 \ldots (n-1) \cdot n = (n-1)! \cdot n \quad \text{for } n \geq 1$$

and $0! = 1$ by definition, we can compute $F(n) = F(n-1) \cdot n$ with the foll. recursive algm.

ALGORITH $F(n)$
// Computes $n!$ recursively
// Input : A nonnegative integer n
// Output : The value of $n!$
if $n = 0$ return 1
Else return $F(n-1) * n$

n - indicates algm. i/p size.

Basic operation of the algm is multiplication. $M(n)$ no. of executⁿ

(or we count the no. of times the comparison $n=0$ is executed, which is the same as counting the total no. of calls made by the algm.).

$F(n)$ is computed acc. to formula,

$$F(n) = F(n-1) \cdot n \quad \text{for } n > 0,$$

the no. of multiplications $M(n)$ needed to compute it must satisfy the equality. E

$$M(n) = \underbrace{M(n-1)}_{\text{to compute } F(n-1)} + \underbrace{1}_{\text{to multiply } F(n-1) \text{ by } n} \quad \text{for } n > 0$$

Thus, the last eqⁿ defines the sequence $M(n)$ that we need to find. This fⁿ defines $M(n)$ not explicitly as a f of n but implicitly as a fⁿ of its value at another point, namely $n-1$. Such equations are called Recurrence relations or, for brevity, recurrences

Solve Recurrence relation $M(n) = M(n-1) + 1$, i.e. to find an explicit formula for $M(n)$ in terms of n only.

To determine solution uniquely we need an initial condition that tells us the value with which the sequence starts. (We can obtain this value by inspecting the condition that makes the algm stop its recursive calls:

if $n=0$ return 1.

1. Since the calls stop when $n=0$, smallest value of n for which the algm is executed & hence $M(n)$ defined is 0.
2. By inspecting pseudocode's exiting lines, we can see that $n=0$, the algm performs no multiplications.

∴ the initial condition is

$$M(0) = 0$$

the calls stop when $n=0$] [no multiplications when $n=0$

~~Thus, we~~ $M(n) = M(n-1) + 1$ for $n > 0$, — ①

$M(0) = 0$

→ Recurrence Relation

A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. To solve a recurrence relation means to obtain a function defined on the natural nos. that satisfy the recurrence.

For Ex: Worst Case Running Time $T(n)$ of the MERGE SORT procedure is described by the recurrence.

$$T(n) = \Theta(1) \text{ if } n=1$$
$$2T\left(\frac{n}{2}\right) + \Theta(n) \text{ if } n>1$$

4 methods for Solving Recurrence:
1. Substitution Method 2. Iteration 3. Recursion Tree 4. Master method

1. Substitution Method:

Consists of 2 main (things) steps:

a. Guess the Solution.

b. Use the mathematical induction to find the boundary Condition & Shows that guess is Correct.

Ex: 1 Solve the Eqⁿ by Substitution Method.

$T(n) = T\left(\frac{n}{2}\right) + n$

We have to show that ~~for some constant c~~ it is asymptotically bound by $O(\log n)$.

For $T(n) = O \log n$ — for some constant c

$T(n) \le c \log n$

↳ In Recurrence Equation

$T(n) \le c \log\left(\frac{n}{2}\right) + 1$

$\le c \log\left(\frac{n}{2}\right) + 1 = c \log n - c \log_2 2 + 1$

$\le c \log n$ for $c \geq 1$

Thus $T(n) = O \log n$

Ex 2: $T(n) = 2T\left(\frac{n}{2}\right) + n \quad n > 1$, Find an Asymptotic bound on T.

Solution:

We guess the Solⁿ is $O(n(\log n))$. Thus for Constant 'c'.

$T(n) \le c \; n \log n$ Put this in given recurrence rel

Now.

$T(n) \le 2c\left(\frac{n}{2}\right) \log\left(\frac{n}{2}\right) + n$

$\le c n \log n - c n \log 2 + n$

$= c n \log n - n (c \log 2 - 1)$

$\le c n \log n$ for $(c \geq 1)$

Thus $T(n) = O(n \log n)$