S. Shahana

UNIT - II

Mathematical Analysis Of Non-Recursive Algorithm

General plan for Analyzing Efficiency of Non-Recursive Algorithms :-

*). Decide on input's size.

*). Identify the algorithm's basic operation.

*). Check whether the number of times the basic operation is executed depends on the size of input..

*). Set up a sum expressing the no. of times the algorithm basic operation is executed.

*). Manipulate the sum using standard rules and formulas.

Two Basic Rules of sum maniputions.

R_1

$$\sum_{i=l}^{u} ca_i = c \sum_{i=l}^{u} a_i$$

R_2

$$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

Two Summation formulas :

S1 $$\sum_{i=l}^{u} 1 = u - l + 1$$

Where $l \leq u$ are some lower and upper integer limits.

S2 $$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1+2+3+\dots+n$$

$$= \frac{n(n+1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

Examples :-

1) To find the largest element in a list of 'n' numbers.

Algorithm :-

// Input : An array A[0...n-1].

// output : Returns the value of Maxval.

Maxval ← A[0]

for i ← 1 to n-1 do

  if A[i] > Maxval

    Maxval ← A[i]

return Maxval.

*). input size:

The no. of elements used in this array is n ie, our input size is 'n' here.

*). Basic Operation:

There are two operations in the loop's body. The comparison $A[i] > maxval$ and the assignment $maxval \leftarrow A[i]$. Since the comparison is executed on each repetition of the loop, we should consider the comparison to be the algorithm's basic operation.

*). whatever the input is, the comparison is made for n-1 times. so, it depends on our input size 'n'.

*). $C(n)$ denotes the no. of times this comparison is executed and try to find a formula expressing it as a function of size n.

The algorithm makes one comparison on each exeution of the loop, which is repeated for each value of the loop's variable i within the bounds between 1 and n-1.

$$\therefore C(n) = \sum_{i=1}^{n-1} 1.$$ (By applying S_1 formula i repeated n-1 times)

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \in \Theta(n).$$

2). To check whether all the elements in a given array are distinct. (element uniqueness Problem).

Algorithm :.

// input : An array A [0.....n-1]

// output : If the elements are distinct Returns "true" otherwise returns "false".

for i ← 0 to n-2 do

    for j ← i+1 to n-1 do

        { if A[i] = A[j] return false

return true

*) i/p size → 'n' in I/p size is equal to no of element the array.

*) Basic operation → Comparison Since the innermost loop contains a Single operation, we should consider it as the algorithm's basic operation.

*). In this algorithm, the no. of element Comparisons will depend not only on 'n' but also on whethere there are any equal elements in the array and if there are, which array positions they occupy. i.e, the efficiency depends on i/p size and Ordering of data. so, we'll discuss this Pbm ~~with~~ to the worst case.

*). Since one comparison is made for each repetition of the innermost loop and this is also repeated for each value of the outer loop.

$$\therefore C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} [(n-1)-(i+1)+1] = \sum_{i=0}^{n-2} (n-1-i)$$

Based on $\underline{\underline{R_2}}$.

$$= \sum_{i=0}^{n-2} n-1 - \sum_{i=0}^{n-2} i \qquad \boxed{\therefore \underline{\underline{R_2}} = \sum_{i=l}^{u} a_i \pm b_i}$$

$$= \sum_{i=0}^{n-2} n-1 - \frac{(n-2)(n-1)}{2}$$

$$= n-1 \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \qquad \boxed{\therefore R_1 = c \sum_{i=l}^{u} a_i}$$

$$= n-1 \cdot (n-2-0+1) - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{2(n-1)^2 - (n-2)(n-1)}{2}$$

$$= \frac{(n-1)(2n-2-n+2)}{2}$$

$$= \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \; \theta(n^2) \; //$$

In general, This algorithm needs to compare all $n(n-1)/2$ distinct pairs of its n elements in the worst case.

3. Matrix Multiplication:

Algorithm.

//input : 2 n by n matrices A & B

// output : Matrix C = AB.

for i ← 0 to n-1 do

    for j ← 0 to n-1 do

        c[i,j] ← 0.0

        for k ← 0 to n-1 do

            c[i,j] ← c[i,j] + A[i,k] * B[k,j]

return C

*). input size → n by matrix Order.

*). Basic operation → multiplication and addition.

In this algm, we consider multiplication as our basic operation at first and then do for th addition since by counting one we automatically count the other.

Let us set up a sum for th total no. of multiplications M(n) first.

$$\text{ie, } M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n \quad (\because \text{by using } S_1)$$

$$M(n) = \sum_{i=0}^{n-1} n^2 = n^3.$$

No of multiplications depends only on the size of the i/p matrices, we do not have to investigate the best, average & worst case efficiency separately

Running time $T(n)$ of this algm. is measured by,

$$T(n) \approx C_m \cdot M(n).$$
$$= C_m n^3.$$

$|||^{ly}$ for addition,

$$A(n) = n^3.$$
$$T(n) \approx C_a \cdot A(n)$$
$$= C_a \cdot n^3.$$

$$T(n) = C_m \cdot M(n) + C_a \cdot A(n) = (C_m + C_a)n^3.$$

where C_a is the time of one addition and C_m is the time of one multiplication.

4. To find the no. of binary digits in the binary representation of a positive decimal integer.

Algorithm:

// input : integer n.

// output : no. of binary digits in n's binary representation.

```
count ← 1
while n > 1 do
      count ← count + 1
      n ← ⌊n/2⌋
return count.
```

In here, the no. of times the comparison will be executed is larger than the no. of repetitions of the loop's body by exactly 1. $\therefore$ we have to use an alternative way of computi

ie, No. of times the comparison is executed = No. of times the Assignment is executed + 1

No. of time the Assign. is executed = $\log_2 n$ since, the value of n is ~~bot~~ about halved on each repetion of the loop.

$\therefore$ No. of times the comparison is exec = $\log_2 n + 1$.

By this, we conclude that whatever the instrs it divides the i/p size by half will have no. of times of execution as $\underline{\underline{\log_2 n}}$.

$\because \lfloor \log_2 n \rfloor + 1 \Rightarrow$ represents the no. of bits in n's binary repn.

Mathematical Analysis of Recursive Algorithms:

General plan for Analyzing efficiency of Recursive Algms:

- Decide an i/p size.
- Identify the Basic Operation.
- check whether $C(n) \in n$ (i/p size) or not.
- Set up a recurrence relation. with an initial condition, for the no. of times the basic operation is executed.
- Solve the recurrence.

Examples :-

1). To find the factorial function $F(n) = n!$

Algorithm :

// input : A nonnegative integer n

// output : Value of $n!$

if $n=0$ return 1

else return $F(n-1) * n$

$$n! = \underbrace{1 \ldots (n-1)}\cdot n$$

$$= (n-1)! \cdot n \qquad \forall n \geq 1$$

$$\therefore \underline{F(n) = F(n-1) \cdot n}$$

i/p size $\rightarrow n$.

Basic operation $\rightarrow$ multiplication.

To compute the no. of multiplications $M(n)$, ~~an~~ it must satisfy the equality

$$\underline{M(n) = M(n-1) + 1} \qquad \forall n > 0$$

$M(n-1)$ for computing $F(n-1)$ and 1 for multiplying the value of $F(n-1)$ by n.

The above equation defines. ~~M(n) not wri~~ we should not write $M(n)$ as a direct function of n (explicitly) ~~by~~ but as a function of its value at another point (implicitly) namely $n-1$. Such equations are called recurrence relation.

finding initial condition:

To determine the solution uniquely we need an initial condition. that tells us the value with which the sequence starts.

from the algm, if ~~no~~ $n=0$ return 1.

So, when $n=0$, the algorithm performs no multiplication.

Thus the initial condition is,

$$M(n) = M(n-1)+1 \quad \text{for } n>0$$

$$M(0) = 0$$

By applying the method of backward substitutions we get the above equation as

$$M(n) = M(n-1)+1$$
$$= [M(n-2)+1]+1 = M(n-2)+2$$
$$= [M(n-3)+1]+2 = M(n-3)+3.$$

In general $= M(n-i)+i$

$\therefore M(n) = M(n-1)+1 = \cdots = M(n-i)+i = \cdots = M(n-n)+n = n$.

By substituting the value $i=n$ in the formula we can get the above result.

2. Towers of Hanoi Puzzle:

→ In this puzzle, we have n disks of different sizes and three pegs. Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on top. The goal is to move all the disks to the third peg, using the second one as auxiliary. ~~But~~ we can move only one disk at a time and never placing a disk on top of the smaller one.

*) i/p size → n.

*) Basic operation → moving a disk.

*) The no. of moves $M(n)$ depends on n only.

*). We get the recurrence equation as,

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$$

finding initial condition:

$$M(1) = 1.$$

$$M(n) = 2M(n-1) + 1 \quad \text{for } n > 1.$$

$$M(1) = 2M(1-1) + 1 = 1.$$

Applying Backward substitutions.

$$M(n) = 2M(n-1) + 1$$

$$= 2[2M(n-2)+1] + 1 = 2^2 M(n-2) + 2 + 1$$

$$= 2^2 [2(M(n-3)+1] + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1$$

In general, $M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \dots + 2 + 1$

$$= 2^i M(n-i) + 2^i - 1.$$

Since the initial condition is $n=1$, we apply the value $i = n-1$ in the above formula.

$$M(n) = 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1$$

$$= 2^{n-1} M(1) + 2^{n-1} - 1$$

$$= 2^{n-1} - 1 = \underline{\underline{2^{n} - 1}}.$$

Algorithm:

Hanoi(no.of disks , start , destination , auxilary).

if (no. of disks > 0)

Hanoi (no. of disks -1, start, auxilary, destination);

Move top disk from source to dest.

Hanoi (no. of disks -1, auxilary, dest, start);

return.

3). To find the no. of binary digit in n's binary repn.

Algorithm:

// input : n

// output : no. of bits in n's binary repn.

if n=1 return 1

else return BinRec($\lfloor n/2 \rfloor$)+1.

i/p size → n

Basic operation → addition.

⑦

*) I/P size : n

*) Basic Operation : Addition.

*). Set up a recurrence equation based on, the no. of additions made in the algorithm is, $A(n) = A(\lfloor n/2 \rfloor) + 1 \quad \forall n > 1$ —— ①

$A(\lfloor n/2 \rfloor) \rightarrow$ no. of additions made in computing BinRec($\lfloor n/2 \rfloor$)

$1 \rightarrow$ to increase the returned value by 1.

*). finding initial conditions :

from the algm, when n is equal to 1, there are no additions Performed. so, the initial condition is, $A(1) = 0$ —— ②

In floor fn we can't apply Backward Subs.

Based on Smoothness rule we assume the variable n takes the value as 2^k which gives the correct answer for all values of n. Since ~~the~~ the

$$① \rightarrow \therefore A(2^k) = A(2^k/2) + 1$$

$$= A(2^{k-1}) + 1 \quad \text{for } k > 0.$$

$$② \rightarrow A(2^0) = 0.$$

Now apply the backward substitutions.

$$A(2^k) = A(2^{k-1}) + 1$$

$$= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2$$

$$= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3$$

In general,

$$A(2^k) = A(2^{k-i}) + i$$

$$\vdots$$

$$= A(2^{k-k}) + k$$

Thus, $A(2^k) = A(1) + k = k$.

after returning to the original variable $n = 2^k$

$$k = \log_2 n$$

$$\therefore \underline{\underline{A(n) = \log_2 n}} \in \Theta(\log n).$$

Fibonacci Numbers :-

0, 1, 1, 2, 3, 5, 8, 13 ....

the recurrence equation for this one is,

$$F(n) = F(n-1) + F(n-2) \quad \forall n > 1 \text{ ——— } (1)$$

initial conditions are, $F(0) = 0$ & $F(1) = 1$. ——— (2)

for computing the above two equations, first we find an explicit formula based on the homogeneous second order linear recurrence with constant coefficients.

general eqn. is,

$$ax(n) + bx(n-1) + cx(n-2) = 0$$

Its characteristic equation is,

$$ar^2 + br + c = 0.$$

where a, b & c are the coefficients of the recurrence.

Let us apply this to the 1 eqn. and we get the eqn as

$$F(n) - F(n-1) - F(n-2) = 0 \quad \text{———} \quad (3)$$

characteristic eqn. is,

$$r^2 - r - 1 = 0 \quad \text{———} \quad (4)$$

solving (4) we get $r_{1,2} = \dfrac{1 \pm \sqrt{1 - 4(1)(-1)}}{2} = \dfrac{1 \pm \sqrt{5}}{2}$

It has two distinct ~~no~~ real roots

so, we need to introduce the new formula $\left(\because \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}\right)$

ie, $F(n) = \alpha \left(\dfrac{1+\sqrt{5}}{2}\right)^n + \beta \left(\dfrac{1-\sqrt{5}}{2}\right)^n$ ——— (5)

when we apply the initial values to the (5)th eqn.

$$F(0) = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^0 + \beta \left(\frac{1-\sqrt{5}}{2}\right)^0 = \alpha + \beta = 0.$$

$$F(1) = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^1 + \beta \left(\frac{1-\sqrt{5}}{2}\right)^1 = \frac{1+\sqrt{5}}{2}\alpha + \frac{1-\sqrt{5}}{2}\beta = 1$$

ie,
$$\boxed{\begin{aligned} &\alpha + \beta = 0 \\ &\frac{1+\sqrt{5}}{2}\alpha + \frac{1-\sqrt{5}}{2}\beta = 1 \end{aligned}}$$
——— I
——— II

from (I). $\beta = -\alpha$ or $\alpha = -\beta$.

substitute $\beta = -\alpha$ in II

$$\left(\frac{1+\sqrt{5}}{2}\right)\alpha - \left(\frac{1-\sqrt{5}}{2}\right)\alpha = 1.$$

$$\frac{\cancel{\alpha}}{\cancel{2}} + \frac{\sqrt{5}\alpha}{2} - \frac{\cancel{\alpha}}{\cancel{2}} + \frac{\sqrt{5}}{2}\alpha = 1$$

$$\frac{\cancel{2}\sqrt{5}\alpha}{\cancel{2}} = 1$$

$$\boxed{\begin{aligned}\alpha &= \frac{1}{\sqrt{5}} \\ \beta &= -\frac{1}{\sqrt{5}}\end{aligned}}$$

Substitute α and β values into the 5th eqn, we get

$$F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Introducing some arbitrary integer power of irrational number ~~assume~~ $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ & $\hat{\phi} = -1/\phi \approx -0.618$; the above eqn should be replaced as,

$$F(n) = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right).$$

The order of growth of this eqn belongs to $\Theta(\phi^n)$ ie, $F(n) \in \Theta(\phi^n)$ hence, $\hat{\phi}$ is between -1 and 0 so it gets infinitely small as n goes to infinity. So, the value of $F(n)$ can be obtained by rounding off the 1st term to the nearest integer. ie, $F(n) = \frac{1}{\sqrt{5}}\phi^n \in \Theta(\phi^n)$.

Algorithms for computing fibonacci numbers.

Algorithm:

// input ; A nonnegative integer n

// output : nth fibonacci no.

if $n \leq 1$ return n

else return $F(n-1) + F(n-2)$

*) i/p size is n.

*) Basic operation is addition.

*) no. of additions needed for computing $F(n-1)$ & $F(n-2)$ are $A(n-1)$ and $A(n-2)$ and one more addition to compute their sum.

Recurrence relation is $A(n) = A(n-1) + A(n-2) + 1 \quad \forall n > 1$

*). Initial Conditions : $A(0) = 0$.

$A(1) = 0$

Solution: $A(n) = A(n-1) + A(n-2) + 1$:

$$A(n) - A(n-1) - A(n-2) - 1 = 0.$$

To eliminate the constant term add 1 to the above eqn.

$$A(n)+1 - A(n-1)+1 - A(n-2)+1 \cancel{-1+1} = 0.$$

$$A(n)+1 - A(n-1)+1 - A(n-2)+1 = 0.$$

assaign $B(n) = A(n)+1$.

$$B(n) - B(n-1) - B(n-2) = 0.$$

ie, $1 + A(n-1) = B(n-1)$

$1 + A(n-2) = B(n-2)$

This eqn is in the form of $ax(n)+bx(n-1)+cx(n-2)=0$,

$\therefore$ $a=1$.

$b=-1$

$c=-1$.

and initial conditions, $A(0)=0 \implies B(0)=0+1=1$

$A(1)=0 \implies B(1)=0+1=1$.

$B(n)=A(n)+1$ ie, it runs one step ahead of $F(n)$

$\therefore$ We can write $B(n)=F(n+1)$.

ie, $A(n)=B(n)-1=F(n+1)-1$

$=\frac{1}{\sqrt{5}}(\phi^{n+1}-\hat{\phi}^{n+1})-1$

thus $F(n)=\frac{1}{\sqrt{5}}\phi^{n}$ and $A(n)\in\Theta(\phi^{n})$.

Computes the nth fibonacci no. iteratively by using an Algo.

Algo:

// input : integer n

// output : nth fibonacci no.

F[0] ← 0;

F[1] ← 1;

for i ← 2 to n do.

F[i] ← F[i-1] + F[i-2]

return F[n]