Mathematical analysis for recursive Algorithm :-

Procedure for Analysing efficiency of recursive algorithm :-

-> Identify the input size 'n'

-> Identify the basic operation of an algorithm

-> Count the no of times basic operation executed

->Identify the relationship between input size and the no of count execution of basic operation.

-> calculate worst case, best case and average case Complexity of an algorithm.

->set up the recurrence operation with its initial condition which expresses basic operation.

->Solve the recurrence equation using forward and backward substitution.

-> prove correctness of an algorithm by using some mathematical induction method.

Example:

Problem Statement:- Compute factorial of 'n' number with recursion

Algo:- To compute n! using recursion.

I/P:- A non-negative integer 'n'

o/p:- if (n=0) return 1 else return factorial (n-1)!*n.

usually, n! can be obtained by repeated multiplication.

If n=5 then

i) n! = 5! = (n-1)!*n => (4)! *5

=> 3! * 4 * 5

=> 2! * 3 × 4 * 5

=> 1! * 2 * 3 * 4 * 5

=> 0! * 1 × 2 × 3 × 4 × 5

=> 1 × 1 × 2 × 3 × 4 × 5.

=> 120.

Analysis

-> I/p size is 'n'

-> Basic operation is multiplication for finding factorial.

-> here recursive call is done as

$$F(n) = F(n-1)! * n, \quad n > 0$$

Take basic operation as M(n), then it is the multiplication count for computing n factorial.

=> M(n) = M(n-1) + 1 ---> to multiply F(n-1) by n.

↓

to compute F(n-1)

Forward Substitution:-

If n=1 M(n) = M(n-1)+1

M(1) = 0+1 => 1

If n=2 M(2) = M(1)+1

= 1+1 => 2

If n=3 M(3) = M(2)+1

= 2+1 => 3.

Backward substitution

$M(n) = M(n-1)+1 \rightarrow (1)$

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

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

Sub (2) in (1)

we get

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

$= M(n-2)+2$

Next

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

$M(n) = M(n-2)+2 \rightarrow (3)$

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

$= M(n-3)+1 \rightarrow (4)$

$M(n) = M(n-3)+3$ by substituting (4) in (3)

hence we can obtain a formula

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

hence complexity can be calculated by as $\Theta(n)$

Mathematical Analysis of Non-Recursive algorithm.

→ Decide on the input size on parameter 'n'

→ Identify algorithm's basic operation.

→ Identify how many times, basic operation is executed

→ Check whether basic operation depends on the input size.

→ Calculate the worst, best and average case complexities for input size 'n'.

→ have to analyze the three complexities in seperate manner.

→ set up a sum for the no. of times the basic op is executed

→ simplify the sum using standard formula and rules.

Ex: Find max element in the given array.

Algo:

// Description :- to find largest element in the given array.

I/p - Array A[0... n-1] of any size 'n'

o/p :- returns the largest element.

```
max ← A[0]
for i ← 1 to n-1 do
{
  if A[i] > max then
    max ← A[i]
}
return max;
```

1, 3, 8, 9, 12
max max max
① ② ③

A[0] = 1
A[1] = 3
A[2] = 8
A[3] = 9
A[4] = 12

A[0] > A[0] => 1 ≥ 1
A[1] > A[0] => 3 > 1 return 3
A[2] > A[0] => 8 > 3 max = 8
A[3] > A[0] => 9 > 8 max = 9
A[4] > A[0] => 12 > 9 return 9

Mathematical analysis

→ i/p size is 'n'.

→ Basic operation - executing loop to finding max value.

→ comparison is done on each execution of 'for loop' and for each value of 'n'.

→ No need of calculating average, best and worst case complexities.

→ C(n) - no of times comparison is done.

→ for (i=1 to n) => n times comparison is made.

$$C(n) = \sum_{i=1}^{n-1} 1$$ ⇒ one comparison is done for each value of 'n'.

→ Simplification ⇒ $C(n) \in n-1 \in \underline{\underline{\Theta(n)}}$.

Pblm 2 :-

getting matrix multiplication of two matrices.

Algo: to do multiplication of 2 matrices.

I/p: Two matrices 'm' and 'n'

O/p: matric 'c' resultant matrix.

Algo routine:

```
for i=0 to n-1 do
    for j=0 to n-1 do
        c[i,j] <- 0
        for k=0 to n-1 do
            c[i,j] = c[i,j] + A[i,k] * B[k,j]
return c;
```

Mathematical Analysis:

-> I/p size = 'n'

-> Basic operation

```
c[i,j] = c[i,j] + A[i,k] * B[k,j]
```

-> here we have to take both addition and multiplication as both basic operations.

-> loop has both operation

-> we can take multiplication as basic operation.

-> we can simplify the basic operation as

= [For loop i] x [For loop j] x [For loop k]

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

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n \quad \left(\sum_{k=0}^{n-1} = n \; \sum_{j=0}^{n-1} = n \right) \text{- hence } n^2$$

$$= \sum_{i=0}^{n-1} n^2 \Rightarrow n^3.$$

Hence time complexity = $\Theta(n^3)$.

Ex:-3

Linear Search

-> used to search element one by one in sequential manner.

Pblm

0	1	2	3	4
7	3	1	2	17

check whether A[0], if not match increment the pointer to A[1] and so on..

x=17 17 ≠ 7

next 3 ≠ 17, next 1 ≠ 17. next 2 ≠ 17 and last 17=17.

hence element 17 is found.

Algo:- for (i ← 0 to n-1) do

if (A[i] == x] then

return i

end if

end.

Best case:- Search element is at first.

Time complexity $\Theta(1)$. $\Omega(n)$.

Average case:- Search element occurs at i^{th} position. P/n.

avg:- $C_{avg}(n) = n+1/2$. $\Theta(n)$

worst case:- Search element is at last. $O(n)$.