Mathematical analysis:

Step 1: The factorial algorithm works for I/p size n.

Step 2: The basic operation in computing factorial is multiplication.

Step 3: The recursive function call can be formulated as

$$f(n) = f(n-1) * n \quad \text{where } n>0$$

Then the basic operation multiplications is given as $M(n)$. And $M(n)$ is multiplication count to compute factorial(n)

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

These multiplications are required to compute factorial (n-1)

To multiply factorial (n-1) by n.

Step 4: In Step 3 the recurrence relation is obtained.

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

Now we will solve recurrence using

- forward substitution:

$$M(1) = M(0) + 1$$
$$M(2) = M(1) + 1 = 1+1 = 2$$
$$M(3) = M(2) + 1 = 2+1 = 3$$

- Backward Substitution:

$$M(n) = \underbrace{M(n-1)}_{\downarrow} + 1$$
$$= \left[\underbrace{M(n-2)}_{} + 1\right] + 1 = M(n-2) + 2$$

From the substitution methods we can establish a general formula as:

$$\boxed{M(n) = M(n-i) + i}$$

Now let us prove correctness of this formula using mathematical induction as follows.

prove $M(n) = n$ by using mathematical induction.

Basis: Let $n = 0$ then

$M(n) = 0$

i.e. $M(0) = 0 = n$

Induction: If we assume $M(n-1) = n-1$ then

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

$\therefore M(n) = n$

Thus the time complexity of factorial function is $O(n)$

Divide and conquer:- overcome, defeat

An algorithmic strategy in which the big problem is broken down into smaller subproblems & solution to these subproblems is obtained.

eg:- binary search, Quick ~~search~~ sort, merge sort.

General method:-

- In divide & conquer method, a given problem is
  i) Divide into smaller sub problems
  ii) ...

Summation formula & Rules used in efficiency analysis (4)

$$\rightarrow \sum_{i=1}^{n} 1 = 1+1+1+\cdots+1 = n \in \Theta(n)$$

$$\rightarrow \sum_{i=1}^{n} i = 1+2+\cdots+n = \frac{n(n+1)}{2} \in \Theta(n^2)$$

$$\rightarrow \sum_{i=1}^{n} i^k = 1+2^k+3^k+\cdots+n^k \approx \frac{n^{k+1}}{k+1} \in \Theta(n^{k+1})$$

$$\rightarrow \sum_{i=1}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1} \in \Theta(a^n)$$

$$\rightarrow \sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$$

$$\rightarrow \sum_{i=1}^{n} c a_i = c \sum_{i=1}^{n} a_i$$

$\rightarrow \sum_{i=k}^{n} 1 = n-k+1$, where n & k are upper and lower limits

- Procedure for control abstraction for D C

Algorithm DC(P)

{

if P is too small then

return solution of P

{

Divide (P) & obtain $P_1, P_2 \ldots P_n$

where $n \geq 1$

Apply DC to each subproblem

return Combine (DC(P_1), DC(P_2) ... (DC(P_n))

}

}

- computing time of DC → recurrence r/s

→ computing time for DC in size n

$$T(n) = \begin{cases} g(n) \\ T(n_1) + T(n_2) + \cdots T(n_r) + f(n) \end{cases}$$

↓ time for DC for i/p size n

↓ Time req for sub problem

efficiency analysis of DC

Let recurrence relation is

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

consider $a \geq 1$ & $b \geq 2$ assume $n = b^k$ where $k = 1, 2 \ldots$

$$T(b^k) = aT(b^k/b) + f(b^k)$$

$$= \underbrace{aT(b^{k-1})} + f(b^k)$$

$$= a[aT(b^{k-2}) + f(b^{k-1})] + f(b^k)$$

$$= a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

now substituting $T(b^{k-2})$ by using back substitution

$$= a^2[aT(b^{k-3}) + f(b^{k-2})] + af(b^{k-1}) + f(b^k)$$

$$= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

continuing this method we get

$$= a^k T(b^{k-k}) + a^{k-1} f(b^1) + a^{k-2} f(b^2) + \ldots + a^0 f(b^k)$$

$$= [a^k T(1) + a^{k-1} f(b) + a^{k-2} f(b^2) + \ldots + a^0 f(b^k)]$$

This can also be written as

By property of logarithm,

$$a^{\log_b x} = x^{\log_b a}$$

hence we can write a^k as

$$a^k = a^{\log_b n} = n^{\log_b a}$$

we can rewrite the equation

$$T(n) = a^k \left[T(1) + \sum_{j=1}^{k} f(b^j)/a^j \right]$$

$$T(n) = n^{\log_b a} \left[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j \right]$$

$\therefore$ order of growth of $T(n)$ depends upon values of constants a & b & order growth of function $f(n)$