Text:

Solving Recurrences (Decreasing Functions)

In this section, we will look at some recurrences of the form

T (n) = a T (n - b) + (n) where *a >1, b > 1,* and *(n)* is a given function.

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs. Since the sub problems here subtracts some constant from the initial input length, it is called as a decreasing function. We are studying this class of recurrences separately because it will help us understand and memorize the Master's theorem effectively. Let us see and solve some recurrences of such type using iteration and recursion tree methods.

Solve using Iteration method

We will look at some examples and solve some decreasing function recurrences using iteration method.

Example 9.1:

```
void disp (int n)
{
        If(n>0)
        {
                for(i=0;i<n;i++)                    ----------> n+1
                        printf("%d", n);            ----------> n
                disp(n-1);                          ----------> T(n-1)
        }                                                       ----------
}                                                           T(n-1) + 2n + 1
                                                          or T(n-1) + n
```

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

T(n) = *T(n-1) + n* ---------- *(1)*

= *[T((n-1)-1)+ (n-1)] + n* Using equation *(1)* for *n-1*

= *T(n-2) + (n-1) + n*

= *[T(n-3)+ (n-2)] + (n-1) + n* again, using equation *(1)* for *n-2*

= *T(n-3)+ (n-2) + (n-1) + n*

.

.

K times

T(n) = *T(n-k) + (n-(k-1)) + (n-(k-2)) + … + (n-2) + (n-1) + n* ----------*(2)*

Here stopping condition is *T(0) = 1*

=> T(n-k) = T(0)

n-k = 0 => k = n ---------- *(3)*

Using (3) in equation (1), we have

T(n) = T(0) + (n-(n-1)) + (n-(n-2)) + … + (n-2) + (n-1) + n

= 1 + 1 + 2 + … + (n-2) + (n-1) + n

= 1 + n(n+1)/2

=> T(n)= O(n^2)

Example 9.2:

```
void disp (int n)
{
      If(n>0)
      {
            for(i=0;i<n;i*2)                  ----------> log n+1
                  printf("%d", n);            ----------> log n
            disp(n-1);                        ----------> T(n-1)
      }                                                  ----------
}                                                        T(n-1) + log n
```

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + log\ n & n > 0 \end{cases}$$

T(n) = T(n-1) + log n ---------- *(1)*

= [T((n-1)-1)+ log (n-1)] + log n using (1) for *n-1*

= T(n-2) + log (n-1) + log n

= [T(n-3)+ log (n-2)] + log (n-1) + log n again, using (1) for *n-2*

= T(n-3)+ log (n-2) + log (n-1) + log n

.

.

K times

T(n)= T(n-k) + log (n- k+1) + log (n- k+2) + … + log (n-2) + log (n-1) + log n -------(2)

Here stopping condition is *T(0) = 1*

=> T(n-k) = T(0)

n-k = 0 => k = n ---------- *(3)*

Using (3) in equation (1), we have

T(n) = T(0) + log (n-n+1) + log (n- n+2) + … + log (n-2) + log (n-1) + log n

= 1 + log 1 + log 2 + … + log (n-2) + log (n-1) + log n

= 1 + log (1 × 2 × .. × (n-1) × n)

T(n) = 1 + log n!

=> T(n) = O(n log n)

Example 9.3:

```
void disp (int n)
{
    If(n>0)
    {
            printf("%d", n);          ----------> 1
             disp(n-1);               ----------> T(n-1)
             disp(n-1);               ----------> T(n-1)
        }                                        ----------
                                                 2T(n-1) + 1
}
```

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$$

T(n) = *2T(n-1) + 1* ---------- *(1)*

= *2[2T((n-1)-1)+ 1] + 1* Using equation (1) for *n-1*

= 2^2*T(n-2) + 2 + 1*

= 2^2 *[2T(n-3)+ 1] + 2 + 1* again, using equation (1) for *n-2*

= 2^3 *T(n-3) + 2^2+ 2 + 1*

.

.

K times

T(n) = 2^k *T(n-k)+ 2^{k-1}+2^{k-2} +… + 2^2+2+ 1* ----------*(2)*

Here stopping condition is *T(0) = 1*

=> T(n-k) = T(0)

n-k = 0 => k = n ---------- *(3)*

Using (3) in equation (1), we have

T(n) = 2^n *T(0)+ 2^{n-1}+2^{n-2} +… + 2^2+2+ 1*

= 2^n+ 2^n *– 1* *here 2^{n-1}+2^{n-2} +… + 2^2+2+ 1 is a GP series*

= 2^{n+1}*-1*

=> T(n)= O(2^n)

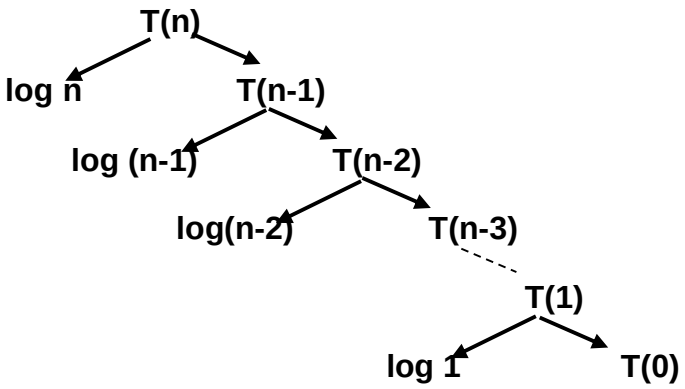
Solve using Recursion Tree method

Let us solve some decreasing function recurrences using recursion tree method.

Example 9.6:

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + \log n & n > 0 \end{cases}$$

Drawing the recurrence tree for *T(n)*, we get



T(n) = *log n + log(n-1) + log (n-2) + … + log 2 + log 1*

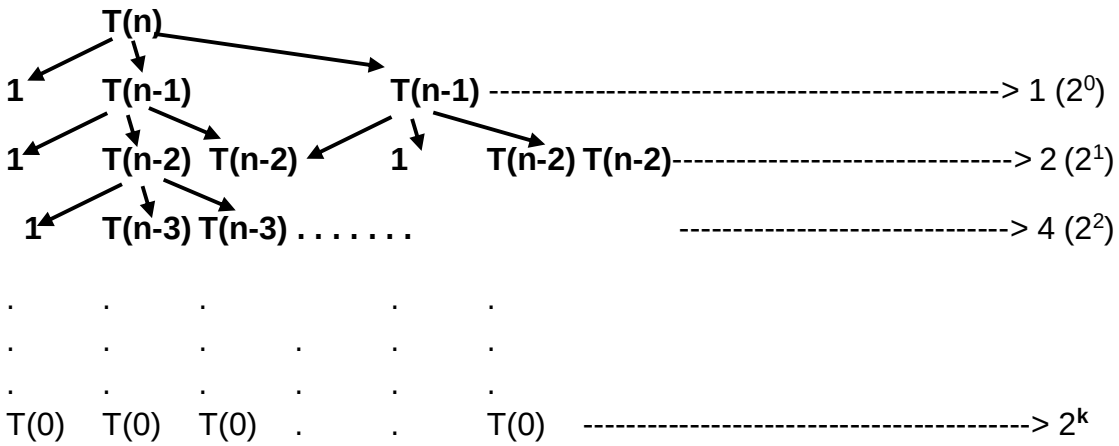
= *log (n × (n-1) × (n-2) × … × 2 × 1)*

= *log n!*

=> *T(n)* = *O (n log n)*

Example 9.7:

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$$



T(n) = *1 + 2 + 4 + 8 + … + 2^k*

The R.H.S a GP series with *a = 1, r = 2*

=> 1 + 2 + 4 + 8 + … + 2^k = 1(2^{k+1}-1)/2-1

T(n) = 2^{k+1} – 1

Here stopping condition is *T(0) = 1*

=> T(n-k) = T(0)

∴, n-k = 0 => k = n

=> T(n) = 2^{n+1} – 1

=> T(n) = O(2^n)