Time Complexity using Recurrence relations:

Some complexity analysis problems can be decomposed into subproblems having a mathematical pattern, which can make use of recurrence relations. A **recurrence relation**, also known as a **difference equation**, is an equation which describes the number of operations T(n) required by an algorithm in terms of a recursive sequence: each term of the sequence is defined as a function of the preceding terms.
Here is an example of a recurrence relation:

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

This can be interpreted as follows:

1. The total number of operations needed for n data items can be obtained by adding one operation to total number of operations needed for n – 1 data items.
2. For a single data item the algorithm requires one operation.

It follows that T(n-1) can also be represented in terms of T(n-2) in similar manner.

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

The right hand side can be reduced in similar manner till we reach T(1), as shown in the lecture class. When we substitute for T(1), we get the solution for T(n).

Let us see how the number of operations for the following code can be expressed in terms of a recurrence relation.

```
j = N;
sum = 0;
while (j > 1) {

    sum = sum + j;
    more = 2 * sum;
    j = j / 2;
}
```

It is noted that the time to solve the problem (number of operations) for the case j=N , is reduced to that of solving the problem for j=N/2, after 3 operations are carried out.

This can be expressed mathematically as

$T(N) = T(N/2) + 3$

It is easy to see the problem for N/2 can be reduced to that of solving the problem for N/4 by using a similar argument.

$$= [\, T(N/4) + 3 \,] + 3$$

$$= T(N/8) + 3\,(3),$$ which can be rewritten as

$$= T(N/2^3) + 3\,(3)$$
$$= T(N/2^4) + 4\,(3)$$
$$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

The pattern is very clear now and one can write the general form as

$$T(N) = T(N/2^k) + k\,(3)$$

Our aim is to reduce the first term on the right hand side to T(1).

Let $2^k = N$, then $k = \log N$
and we can express T(N) as

$$T(N) = T(1) + 3 \log N$$

So the complexity turns out to be $O(\log N)$