###BEGIN LESSON###

C949, the eighth section of the tenth chapter of our textbook (10.8 Analyzing the Time Complexity of Recursive Algorithms) gives the following rule based upon recurrence relationships without a “formal” proof. That is:

IF T(N) = O(1) + T(N/2), THEN T(N) = O(log N).

Simply put, this relationship states that the time required for processing N items is a constant amount of time plus the time for processing N/2 items. This statement starts a cascading pathway. So, the time for processing N/2 items is a constant amount of time plus the time required for handling (N/2)/2 or N/4 items. And, this will continue until the time for the current step is a constant amount of time.

As with any “well-structured”, “in-class” proof; let us first look at three “simple” cases of this problem, determining the appropriate running time for T(N) = O(1) + T(N/2) in terms of Big-O notation. Then, we will examine a “general” proof.

CASE I:

Let the size of the input N be 256.

T(256) = O(1) + T(128)

T(128) = O(1) + T(64)

T(64) = O(1) + T(32)

T(32) = O(1) + T(16)

T(16) = O(1) + T(8)

T(8) = O(1) + T(4)

T(4) = O(1) + T(2)

T(2) = O(1) + T(1)

T(1) = O(1)

Substituting upward produces:

T(256) = O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)

= 8 * O(1) + O(1)

So,

T(256) approximates log(256) * O(1) + O(1)

The logarithm of 256 is exactly 8.

CASE II:

Let the size of the input N be 1024.

T(1024) = O(1) + T(512)

T(512) = O(1) + T(256)

T(256) = O(1) + T(128)

T(128) = O(1) + T(64)

T(64) = O(1) + T(32)

T(32) = O(1) + T(16)

T(16) = O(1) + T(8)

T(8) = O(1) + T(4)

T(4) = O(1) + T(2)

T(2) = O(1) + T(1)

T(1) = O(1)

Substituting upward produces:

T(1024) = O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)

= 10 * O(1) + O(1)

So,

T(1024) approximates log(1024) * O(1) + O(1)

The logarithm of 1024 is exactly 10.

CASE III:

Let the size of the input N be 8192.

T(8192) = O(1) + T(4096)

T(4096) = O(1) + T(2048)

T(2048) = O(1) + T(1024)

T(1024) = O(1) + T(512)

T(512) = O(1) + T(256)

T(256) = O(1) + T(128)

T(128) = O(1) + T(64)

T(64) = O(1) + T(32)

T(32) = O(1) + T(16)

T(16) = O(1) + T(8)

T(8) = O(1) + T(4)

T(4) = O(1) + T(2)

T(2) = O(1) + T(1)

T(1) = O(1)

Substituting upward produces:

T(8192) = O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)

= 13 * O(1) + O(1)

So,

T(8192) approximates log(8192) * O(1) + O(1)

The logarithm of 8192 is exactly 13.

CASE IV:

Let the size of the input N be 2^K. Where 2^K / 2 is 2^{K-1} and $2^{K-K} = 2^0 = 1$.

$T(2^K) = O(1) + T(2^{K-1})$

$T(2^{K-1}) = O(1) + T(2^{K-2})$

$T(2^{K-2}) = O(1) + T(2^{K-3})$

…

$T(2) = O(1) + T(1)$

$T(1) = O(1)$

Substituting upward produces:

$$T(2^K) = O(1) + O(1) + O(1) + \ldots + O(1) + O(1)$$

$$= K * O(1) + O(1)$$

So,

$T(2^K)$ approximates $\log(2^K) * O(1) + O(1)$

The logarithm of 2^K is exactly K.

And a result, this recurrence relationship describing the time complexity of binary search and similar recursive functions that reduce their workspace in half during each iteration, $T(N) = O(1) + T(N/2)$ is the equivalent of $T(N) = O(\log N)$.

Algorithms which use this approach are called divide and conquer procedures.

Plus, the process of systematically determining a procedure’s time complexity is called an approximation algorithm.

Finally, these complexities are described in terms of the "natural" ordering of the growth rate of various classes of functions using a descriptor called “Big-O notation”. Where the “O” stands for "on the ‘order’ of".

###END LESSON###