A recurrence relation is just an equation that's defined recursively. Divide and conquer algorithms work by splitting a problem up recursively into two or more pieces at each level until a base case is reached. The time complexities of divide and conquer algorithms are intrinsically recursive themselves, and as a result can easily be modeled using recurrence relations.

In computer science we tipically denote recurrence relations using this format:

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

- T is the algorithm itself, the one that we're examining;
- a represents the number of recursive calls that we make at each level (this is the number of times that we divide a level);
- b is the rate at which we shrink the problem size (when we make a recursive call, b is the factor that determines how much smaller the problem size is in that recursive call);
- f is the work that's done at a level (this could be the work that it takes to actually split-up the original problem size into these smaller problems, or the work to combine the solutions to the base cases).

So, in a recurrence relation, $aT(n/b)$ represents the total amount of work that's done by all of the base cases, and f represents the work done at any given level.

In order to fully define an algorithm using a recurrence relation, it's also necessary to define base cases, because it is a recursive equation. In computer science the base case is usually 0 or 1.

Many common functions can easily be expressed using recurrence relations. Let's see some examples:

$T(n) = T(n-1) + 1, \quad T(1) = 1 \longrightarrow n;$

$T(n) = 2 \cdot T(n-1), \quad T(1) = 1 \longrightarrow 2^{n-1};$

$T(n) = n \cdot T(n-1), \quad T(1) = 1 \longrightarrow n!$