Recurrence Relations
Recitation 9

1 Definition

A recurrence relation for the sequence $\{ a_n \}$ is an equation that expresses $\{ a_n \}$ in terms of one or more of the previous terms for all integers n with $n \geq n_0$ where n_0 is a nonnegative number.

Translation:

This is simply a recursive function or a function that is defined by itself. The most common example are the Fibonacci numbers: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34,...$ The pattern is $f(n) = f(n-1) + f(n-2)$ where $f(0) = 0, f(1) = 1$. In order to calculate $f(n)$ we must first For example:

$$f(4) = f(4-1) + g(4-2)$$

$$f(4) = f(3) + f(2)$$

In order to solve $f(4)$ we need $f(3)$ *and* $f(2)$:

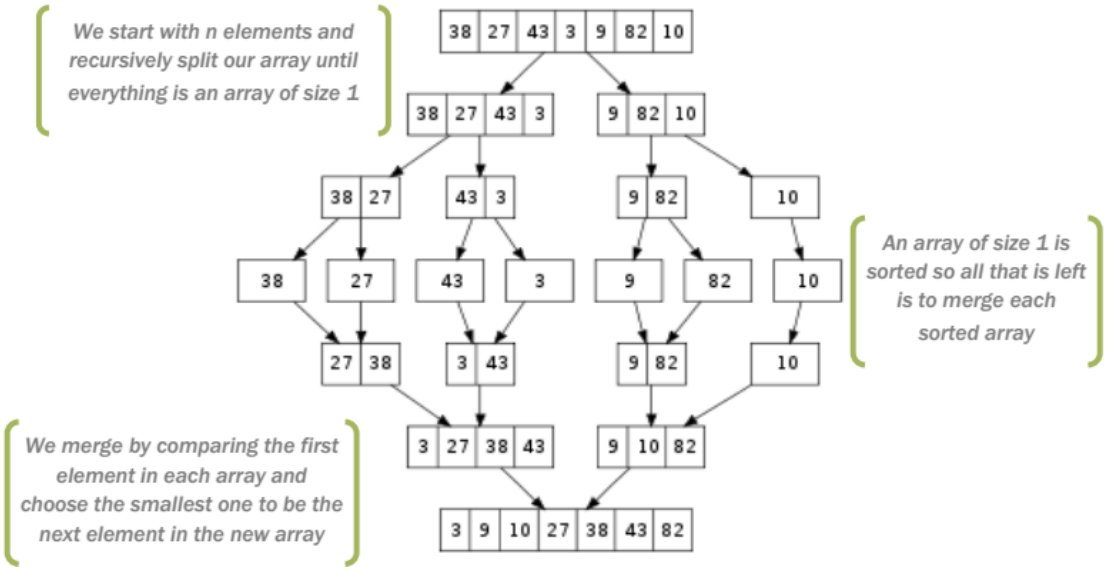
$$f(2) = f(2-1) + f(2-2)$$
$$f(2) = 0 + 1 = 1$$

$$f(3) = f(3-1) + f(3-2)$$
$$f(3) = f(2) + f(1)$$
$$f(3) = 1 + 1 = 2$$

$$f(4) = f(3) + f(2)$$
$$f(4) = 2 + 1 = 3$$

2 Solving with back substitution

Plug the recurrence back into itself over and over until you can see a pattern. Let's do an example with merge sort algorithm.



At each level we recurse twice for each array and each time we recurse on half of the array. This gives us the start of our recurrence relation:

$$f(n) = 2 * f\left(\frac{n}{2}\right)$$

Once we hit out base case, $f(1) = 1$, all that is left is to merge the arrays at each level together. Since at each level there are ultimately still n elements this will require n comparisons. So our final recursive relation for merge sort will look like this:

$$f(n) = 2 * f\left(\frac{n}{2}\right) + n$$

To solve this recurrence relation, we will start plugin f(n) to itself until we can recognize a pattern:

$$f(n) = 2 * f\left(\frac{n}{2}\right) + n$$

$$f(n) = 2 * \left(2 * f\left(\frac{n}{2}\right) + \frac{1}{2}n\right) + n$$

$$f(n) = 4 * f\left(\frac{n}{4}\right) + 2n$$

$$f(n) = 4 * \left(2 * f\left(\frac{n}{8}\right) + \frac{1}{4}n\right) + 2n$$

$$f(n) = 8 * f\left(\frac{n}{8}\right) + 3n$$

$$f(n) = 16 * f\left(\frac{n}{16}\right) + 4n$$

...

There is a pattern that we can recognize here:

$$f(n) = 2^k * f\left(\frac{n}{2^k}\right) + kn$$

So this pattern will continue until we hit our base case of $f(1) = 1$. This implies $\frac{n}{2^k} = 1$, therefore $n = 2^k$, and $k = log_2 n$. This gives us:

$$f(n) = log_2(n) * 1 + log_2(n) * n$$

From here we can also determine a big-O bound on the number of steps in merge sort by finding the big-O of the above function.

$$O(n * log(n))$$