n log n Sorting Algorithms, Reading

It's all about "**scaling**". When data structures contain a small number of values -- when "n" is "small" -- any sorting method works well. Nested-for-loop, insertion sort, bubble sort -- any O(n^2) algorithm works well.

But when n gets large, runtime can become prohibatively long. Here's why: if n is 100 and sorting it takes 1 second, using the same sorting algorithm with n=100,000 takes 12 days! Do the math: 100,000 is 1,000 time more than 100. n^2 is 1,000 x 1,000. 1,000,000 seconds is about 12 days. This is where **n log n algorithms become important** to know.

The Approach

The basic approach for n log n sorting is to use a method that **divides the data set into two** halves, and separate those halves the same way, and so on. That's O(log n) -- like binary search. But it has to be done n times, so that's where n log n comes in.

The Effect

An n log n algorithm that takes 1 second to sort 100 values, takes 2,500 seconds to sort 100,000 values. Do the math: 1 second x (100,000 x log(100,000)) / (100 x log(100)). Way better than 12 days! And it's not very much worse than O(n), which would be 1,000 seconds -- see if you can figure out *that* math.

Limitations

There are algorithms that lend themselves to just **arrays**, just **linked lists**, or to both. The ones for arrays do *not* work well with "**holes**", so use these only for arrays whose "in use" values are all in the left-half and the unused in the right-half of the array, when it's not filled to capacity.