Explanation of Code

Not required
When calling sort, the outer *for loop* is responsible for keeping track of the size of sorted sub-arrays, because the sorted components are logically separated.
A quick peek into the debugger using input array of [3, 5, 1, 6, 7, 2, 8, 4] shows

sortedSize	input array(with brackets to emphasize logical separation)
1	[3], [5], [1], [6], [7], [2], [8], [4]
2	[3, 5], [1, 6], [2, 7], [4, 8]
4	[1, 3, 5, 6], [2, 4, 7, 8]
8*	[1, 2, 3, 4, 5, 6, 7, 8]

*Does not enter loop on sortedSize == 8

The inner *for loop* is responsible for merging adjacent sorted sub-arrays into a single sorted sub-array (or array if it is performing the final merge). Again, because the sub-arrays are only logically separated, it has to ensure it passes in the correct indices into merge.

The merge functions as the same as any merge you've seen before; it takes two sorted arrays and creates one sorted array. leftIndex marks the start point of the **first** sorted sub-array (inclusive). midIndex marks the end point of the **first** sorted sub-array (inclusive). rightIndex marks the end of the **second** sorted sub-array (inclusive).

It is also known as bottom-up merge sort because it begins by treating the elements individually; whereas the recursive version takes the entire array and breaks it down into smaller and smaller pieces. This image illustrates the recursive form.

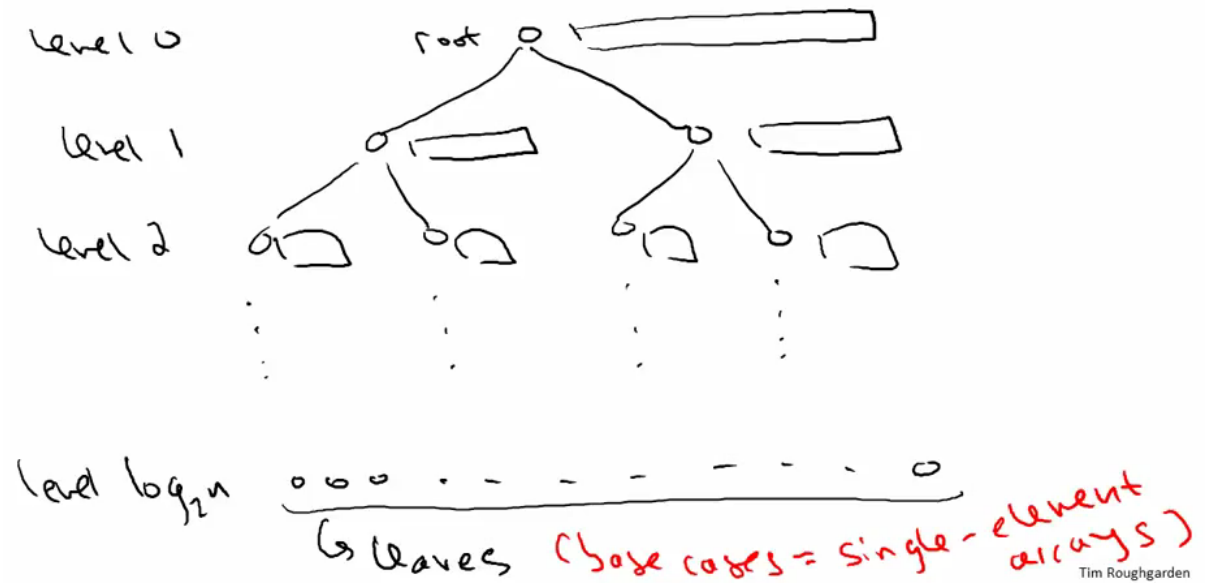

Figure 1

More about top-down/bottom-up http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design

Analysis

Many of you did not include an analysis

A quick note- since we are moving away from recursion and into iteration, it no longer follows the divide-and-conquer paradigm:

- Divide the problem into one or more subproblems.
- Conquer subproblems by solving them **recursively**. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem.

See *http://stackoverflow.com/questions/9126690/compare-divide-and-conquer-with-recursion* for more information.

Let N be the size of the input array. The outer *for loop* loops roughly $log_2(N)$, ignoring constants. The inner for loop is responsible for merging adjacent sub-arrays. Tim Roughgarden showed that merging together arrays at any level uses at most $6N$ operations. Since each iteration requires 6N operations and we have $log_2(N)$ iterations, this gives us a running time of $6Nlog_2(N)$. Expressed as big-O, $O(nlog(n))$. ■