NlogN sorting algorithms

sorting | algorithms | nlogn | time complexity

There are several sorting algorithms; each one has a different best and worst-case time complexity and is optimal for a particular type of data structure. Let's look at some of the sorting algorithms that have a best-case or average-case time complexity of $O(n\ log(n))$.

Merge Sort

Merge sort is a classic example of a *divide-and-conquer* algorithm. It has a guaranteed running time complexity of $O(n \ log(n))$ in the best, average, and worst case. It essentially follows two steps:

1. Divide the unsorted list into sub-lists until there are N sub-lists with one element in each (N is the number of elements in the unsorted list).
2. Merge the sub-lists two at a time to produce a sorted sub-list; repeat this until all the elements are included in a single list.

HeapSort

Heapsort uses the heap data structure for sorting. The largest (or smallest) element is extracted from the heap (in $O(1)$ time), and the rest of the heap is re-arranged such that the next largest (or smallest) element takes $O(logn)$ time. Repeating this over n elements makes the overall time complexity of a heap sort $O(n\, log(n))$.

Quick Sort

Like merge sort, quick sort is a divide-and-conquer algorithm that follows three essential steps:

1. Select an element that is designated as the *pivot* from the array to be sorted.
2. Move smaller elements to the left of the *pivot* and larger elements to the right of the *pivot*.
3. Recursively apply steps 1 and 2 on the sub-arrays.

However, the choice of the pivot actually determines the performance of quicksort. If the first or the last element of the array is chosen as a pivot, then quicksort has a worst-case time complexity of $O(n^2)$. But, if a good pivot is chosen, the time complexity can be as good as $O(n\ log(n))$ with performance exceeding that of merge sort.