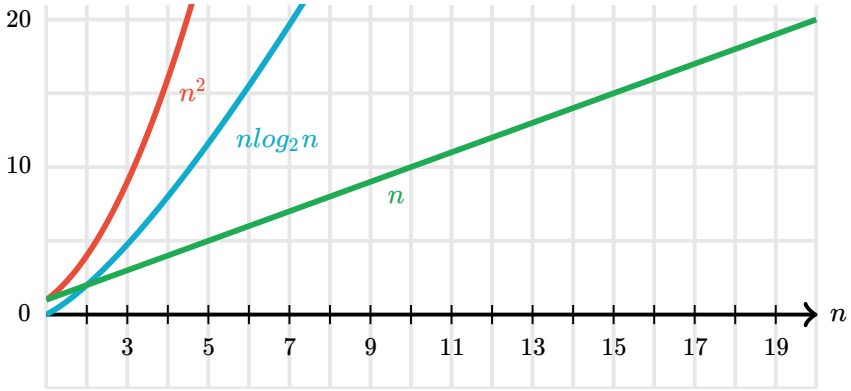
Khan Academy

Divide and conquer algorithms

The two sorting algorithms we've seen so far, selection sort and insertion sort, have worst-case running times of $\Theta(n^2)$. When the size of the input array is large, these algorithms can take a long time to run. In this tutorial and the next one, we'll see two other sorting algorithms, merge sort and quicksort, whose running times are better. In particular, merge sort runs in $\Theta(n \lg n)$ time in all cases, and quicksort runs in $\Theta(n \lg n)$ time in the best case and on average, though its worst-case running time is $\Theta(n^2)$. Here's a table of these four sorting algorithms and their running times:

Algorithm	Worst-case running time	Best-case running time	Average-case running time
Selection sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Insertion sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$

Algorithm	Worst-case running time	Best-case running time	Average-case running time
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$

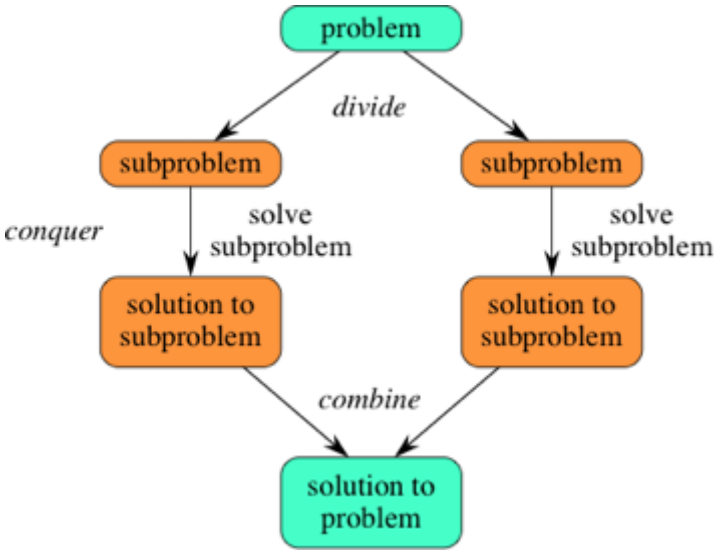


Divide-and-conquer

Both merge sort and quicksort employ a common algorithmic paradigm based on recursion. This paradigm, **divide-and-conquer**, breaks a problem into subproblems that are similar to the original problem, recursively solves the subproblems, and finally combines the solutions to the subproblems to solve the original problem. Because divide-and-conquer solves subproblems recursively, each subproblem must be smaller than the original problem, and there must be a base case for subproblems. You should think of a divide-and-conquer algorithm as having three parts:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

You can easily remember the steps of a divide-and-conquer algorithm as *divide*, *conquer*, *combine*. Here's how to view one step, assuming that each divide step creates two subproblems (though some divide-and-conquer algorithms create more than two):



If we expand out two more recursive steps, it looks like this:

Because divide-and-conquer creates at least two subproblems, a divide-and-conquer algorithm makes multiple recursive calls.

Sort by: Top Voted

Questions Tips & Thanks

Want to join the conversation?

Log in