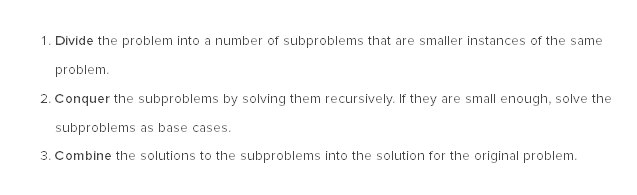
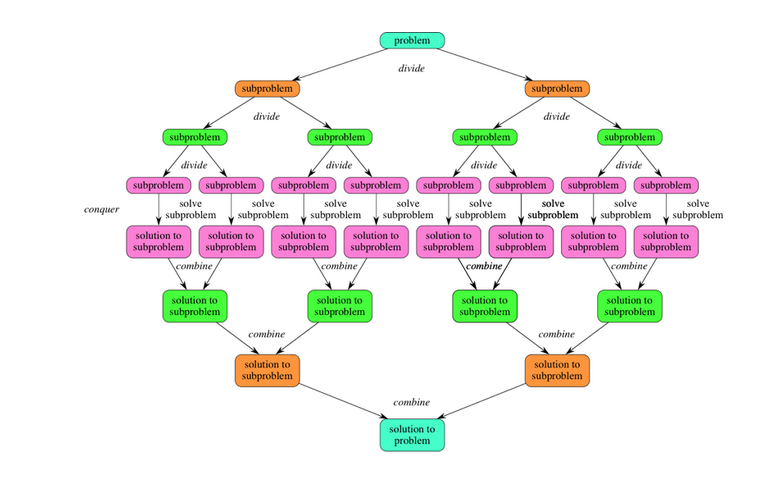
**Divide and conquer algorithms**

Quick sort and merge sort are algorithm that run much better than selection and insertion sort (they have a theta(n\*lg n) instead of theta(n²)

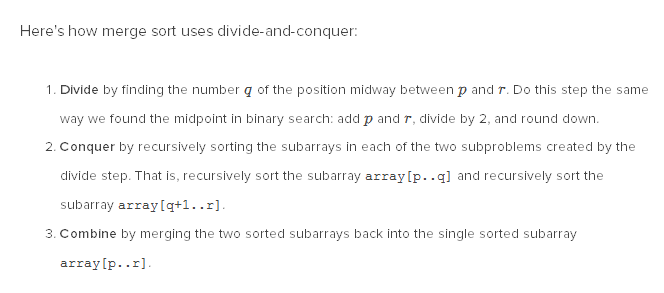
Divide and conquer is a paradigm through which we break a problem into smaller problems and solve them recursively.

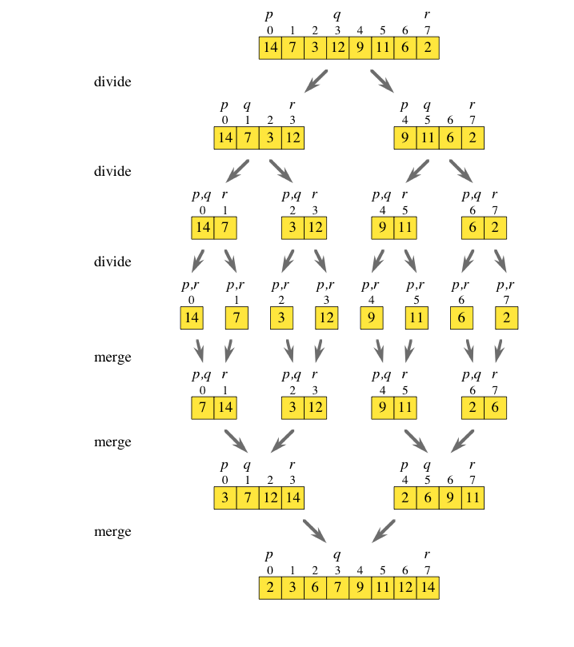


We’re basically going full Cartesian on this. And we divide the subproblems into even smaller subporblems if they prove to be tricky, like so:



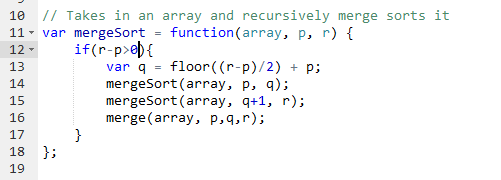
**Merge sort**





The tricky part here seems to be the merge: we are faced with two subarrays that are sorted, but some of them have values that are smaller and bigger than those in the other subarray.

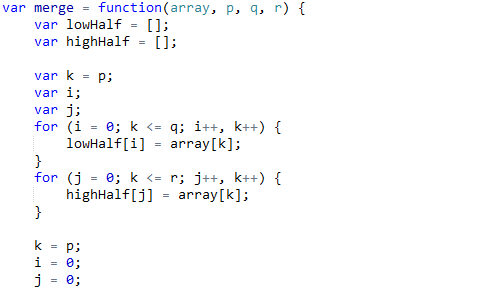
This is how the big loop looks like:



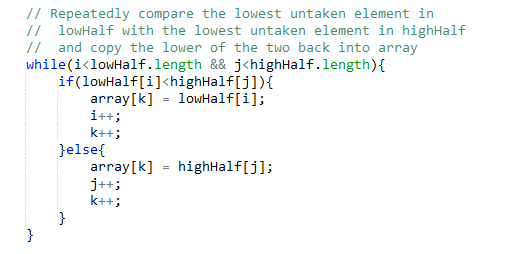
Now for the merge function: this is where the magic happens. It should have, at best a theta(n).

Not that because of recursion, and that we are recursing until the base case (array.length=1), the two subarray we are each time trying to merge are already sorted. So we create a third array, and we compare the first two values of the subarrays. We then increment by one the index of the array that has had the bigger value.

first, we copy the elements left of the middle of the array into a temporary array called lowHalf, and right of the middle into a temporary array called highHalf



Note that we reset i and j because we will need them later to swipe lowHalf and highHalf. Next we are going to cvompare one by one the values of both subarrays, incrementing the index of whichever subarray “wins” the “manche”.



Then we proceed to check in which of the two subarrays we have not yet reach the end of it through our swipe and we copy the remaining values into the big array (they are already sorted):

