Sort an array of *n* numbers.

Establish the problem.

- specifications

Task: sort in increasing (non-decreasing) order

Input: array of *n* numbers

Output: array of *n* numbers, sorted

- examples

Identify avenues of attack.

- targets

brute force: selection sort Th(n^2)

- approach

Divide-and-conquer.

- paradigms and patterns

Paradigm: divide-and-conquer.

Patterns: easy split: split array into first half, second half / friends sort each half / combine two sorted lists

easy merge: split array into smaller things, bigger things / friends sort each half / friends return first half of sorted list (smaller things), second half (bigger things)

Define the algorithm.

- size

n – number of numbers to sort

- generalize / define subproblems

Task: sort A[low..high] (inclusive) in increasing (non-decreasing) order

Input: array A of *n* numbers to sort, range: low, high

Output: A[low..high] is sorted

- base case(s)

n=0: low=high+1 → already sorted! nothing to sort! yay! return

n=1: low=high → already sorted! yay! return

- main case

// split into smaller, bigger

pivot ← A[low]

rearrange elements in A[low..high] so that the first things are < pivot, then the pivot, then > pivot

// hand off to friends

sort(A,low,pivot slot-1) to one friend

sort(A,pivot slot+1,high) to other friend

// magic! friends already sorted it! (if they are sorting within the first part and second part of A[low..high])

- top level
    - initial subproblem

sort(A,0,n-1)

    - setup
    - wrapup
- special cases

n=0 → done! no subproblem

duplicates → don't handle pivot being duplicate – change < to <=

- algorithm

Show termination and correctness.

- termination
    - making progress

argue: why friends always a smaller prob: we don't include the pivot in what the friends get

    - the end is reached

base cases cover 0, 1 element, with n >= 2 then friends get at least 0 elements

- correctness
    - establish the base case(s)

n=0, n=1 → nothing to sort, already sorted

    - show the main case

A[low..high] is arranged into < pivot, pivot, > pivot which holds even when < and > parts are sorted

    - final answer

0..n-1 covers the entire array, so sort(A,0,n-1) sorts the whole array

Determine efficiency.

- implementation
- time and space

T(n) = 2 T(n/2) + ?? – best case

T(n) = T(n-1) + ?? – worst case

- room for improvement