Sort *n* numbers in increasing order.

Establish the problem.

- specifications

task: sort *n* numbers in increasing order

input: array of *n* numbers

output: array of *n* numbers

- examples

Identify avenues of attack.

- known targets

selection, insertion sort – $O(n^2)$

- approach

divide-and-conquer

- paradigms and patterns

easy split – sort first half of the array, sort second half of the array

easy merge – sort smaller numbers, sort larger numbers

Define the algorithm.

(easy split approach)

- generalize / define subproblems

subproblem task: sort part of an array of numbers (A[start..end], inclusive) in increasing order

subproblem input: array A of numbers, range (start..end, inclusive) to sort

subproblem output: array of numbers

- base case(s)

sort 0 or 1 numbers

sort(A,start,end) – if start == end (1 number), it’s sorted!

- main case

idea: easy split – sort first half of the array, sort second half of the array

// split array in half and recursively sort each half

mid ← (start+end)/2

sort(A,start,mid)

sort(A,mid+1,end)

// combine two sorted ranges of an array start..mid and mid+1..end into one start..end

// for simplicity, make a new sorted array B, then copy back over A[start..end]

keep pointers to the smallest unmerged element in each half, respectively, and repeatedly add the smallest of the two currently pointed at elements at the end of B

copy the elements of B into the range A[start..end]

- top level
  - initial subproblem – sort(A,0,n-1)
  - setup – none
  - wrapup – none

- special cases

account for duplicates – does our algorithm deal with that?

- algorithm

Show termination and correctness.

- termination
  - size

n – the number of numbers to sort

  - making progress

...show that the smallest recursive case (2 numbers) splits into 1 and 1...

  - the end is reached

...argue why we don't skip the base case – subproblems are always smaller, all numbers go into a subproblem, so each subproblem has at least 1 element...

- correctness
  - establish the base case(s)

of course 1 number is sorted

  - show the main case

...show we don't lose any numbers – all n that we get go to exactly one friend...

...explain why merging the sorted lists is right...

- final answer

need sort(A,0,n-1) to sort the whole array – subproblem sorts A[start..end] inclusive so A[0..n-1] covers the whole size n array

Determine efficiency.

- implementation
- time and space

T(n) = 2T(n/2) + O(n) → O(n log n), which beats selection and insertion sort

- room for improvement