I) Mergesort

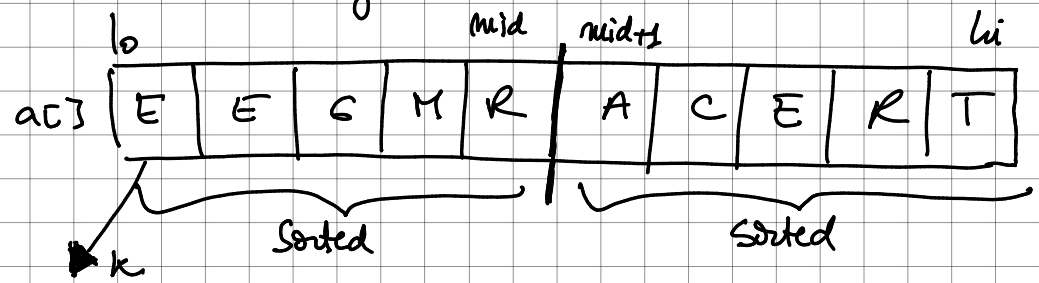
- one of two classic sorting algorithms

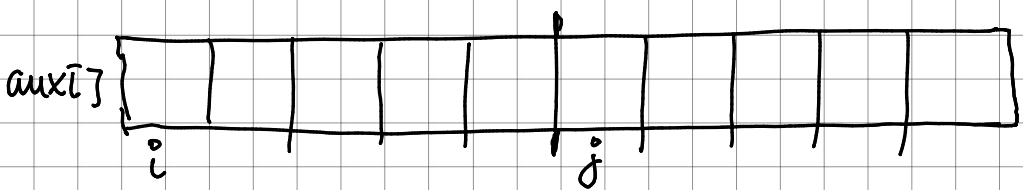
Merge sort - Java sort for objects

Quick sort - Java sort for primitive types

Basic Plan

- divide array in 2 halves
- recursively sort each half and
- merge the two halves





given two sorted subarrays a[lo] - a[mid]
a[mid+1] - a[hi]

replace with sorted sub-array $a[lo] - a[hi]$

- at each step → compare the minimum in each subarray, move that and increment its pointer

---

Proposition: Mergesort uses at most $N \lg N$ compares and $6N \lg N$ array accesses to sort an array of size N

Proof

$C(N)$ – no. of compares

$A(N)$ – no. of array accesses

satisfies the recurrences:

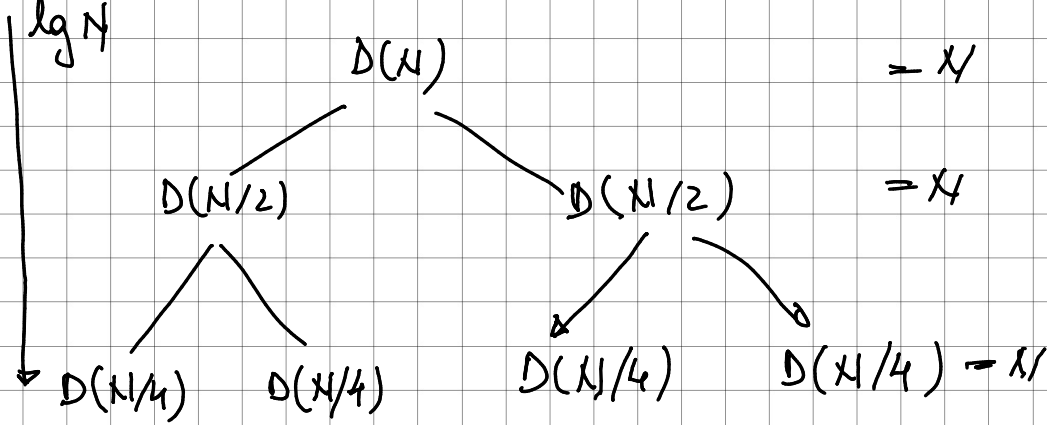
$$C(N) \leq \underbrace{C(N/2)}_{\text{left half}} + \underbrace{C(N/2)}_{\text{right half}} + \underbrace{N}_{\text{merge}}, \quad N > 1$$
$$C(1) = 0$$

$$A(N) \leq A(N/2) + A(N/2) + 6N, \quad N > 1$$
$$A(1) = 0$$

• we solve the recurrence when n is a power of 2 $\Rightarrow$ result holds for all N

$$D(N) = 2D(N/2) + N, \quad N > 1, \quad D(1) = 0$$

$$\Rightarrow \boxed{D(N) = N \lg N}$$



cont $\boxed{N \times \lg N}$

Mergesort: Memory analysis

- extra aux array for the merge operation
- it's not a in-place merge
- possible in theory, too complicated in practice

Optimizations

- use insertion sort for small subarrays
- too much overhead for tiny arrays

* eliminate the copy of the aux array
  - save time not space by switching the role of the input in and auxiliary array in each recursive call

Bottom-up Mergesort

- pass through array; merge subarrays of size 1
- repeat for subarray size of 2, 4, 8 ...