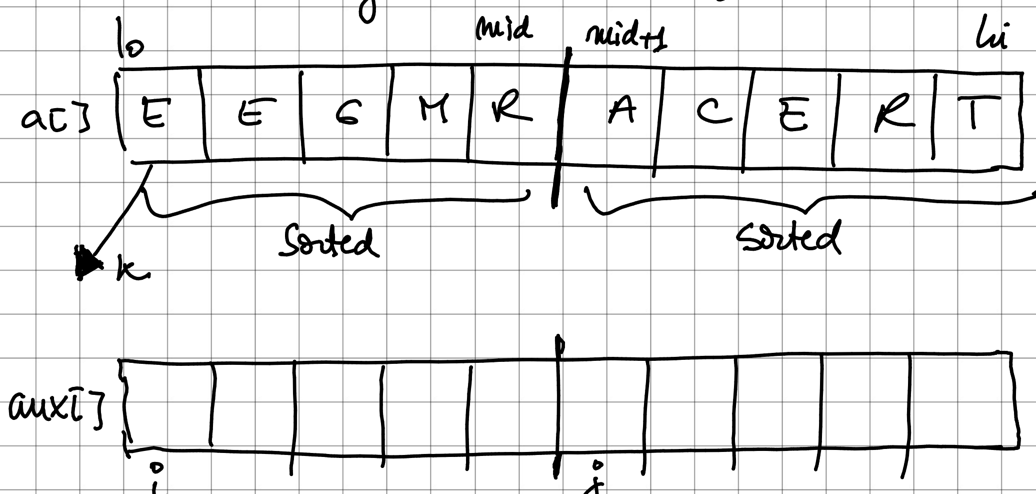
I) Mergesort

- one of two basic 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 —o compare the minimum in each subarray, ~~move~~ copy that and increment its pointer

---

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

<u>Proof</u>

$C(N)$ - no. of compares

$A(N)$ - no. of array accesses

satisfies the recurrences:

$$C(N) \le \underbrace{C(N/2)}_{\text{left half}} + \underbrace{C(N/2)}_{\text{right half}} + \overbrace{N}^{\text{merge}}, \quad N > 1$$

$$C(1) = 0$$

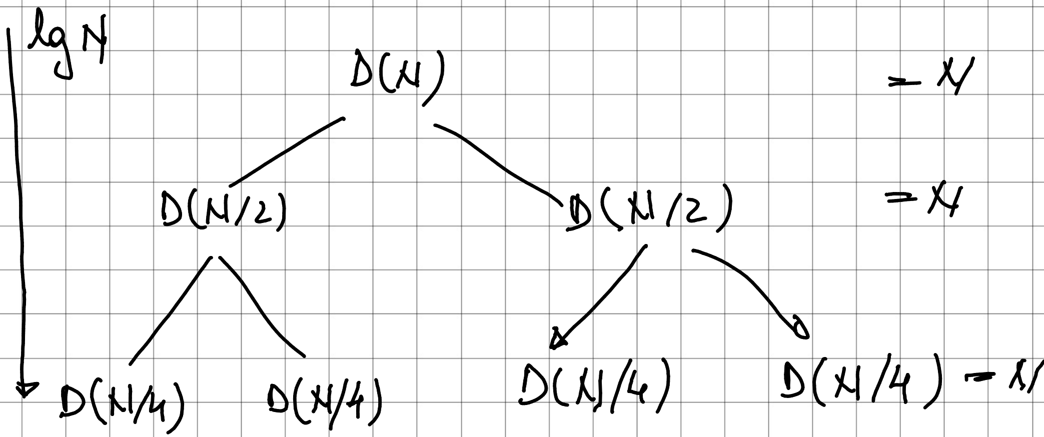
$$A(N) \le A(N/2) + A(N/2) + 6N, \quad N > 1$$

$$A(1) = 0$$

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

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

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



<u>cost</u> $\boxed{N \times \lg N}$

<u>Mergesort: Memory analysis</u>

- extra aux array for the merge operation
- it's not a <u>in-place merge</u>
- 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