I) Mergesort

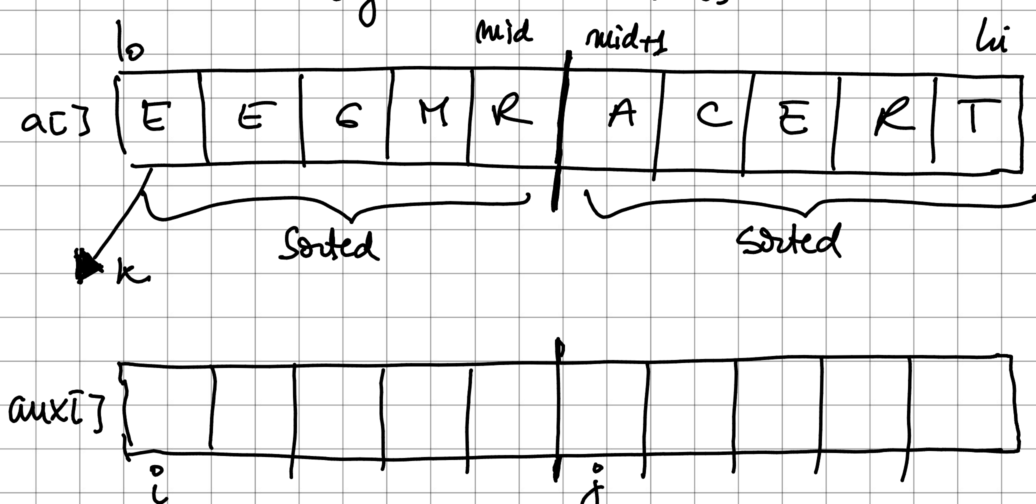
- One of two basic sorting algorithms

<u>Merge sort</u> - Java sort for objects

<u>Quick sort</u> - Java sort for primitive types

<u>Basic Plan</u>

- 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 $\multimap$ 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:

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

$C(1) = 0$

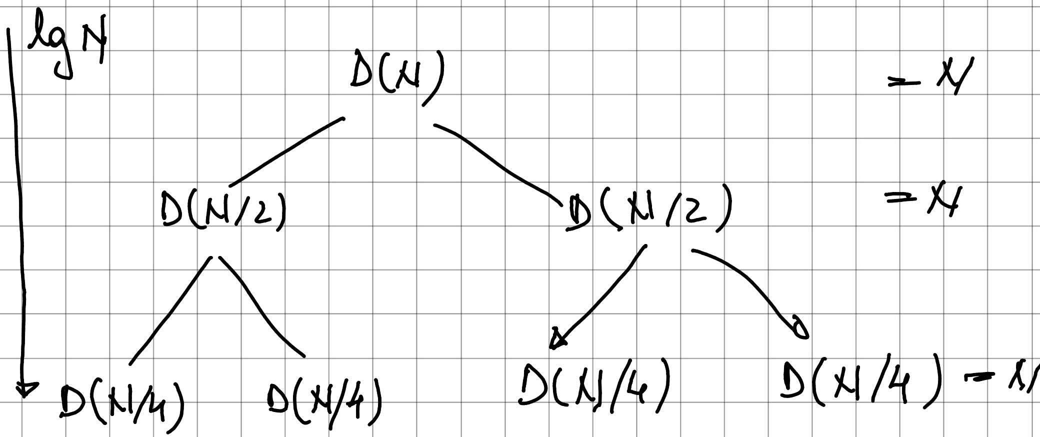
$$A(N) \le A(N/2) + A(N/2) + 6N, \quad N > 1$$

$A(1) = 0$

- 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> $\boxed{N \times \lg N}$

Mergesort: Memory analysis

- 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 ~~and~~ 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 . . .

Complexity of sorting

- computational complexity = framework to study efficiency of algorithms for solving a particular problem X

- model of computation - allowable operations
- cost model: operation count(s)
- upper bound: cost guarantee provided by some algorithm for x
- lower bound: proven limit on cost guarantee of all algorithms for x (no algorithm can do better)
- optimal algorithm - best possible cost guarantee (lower bound ~ upper bound)

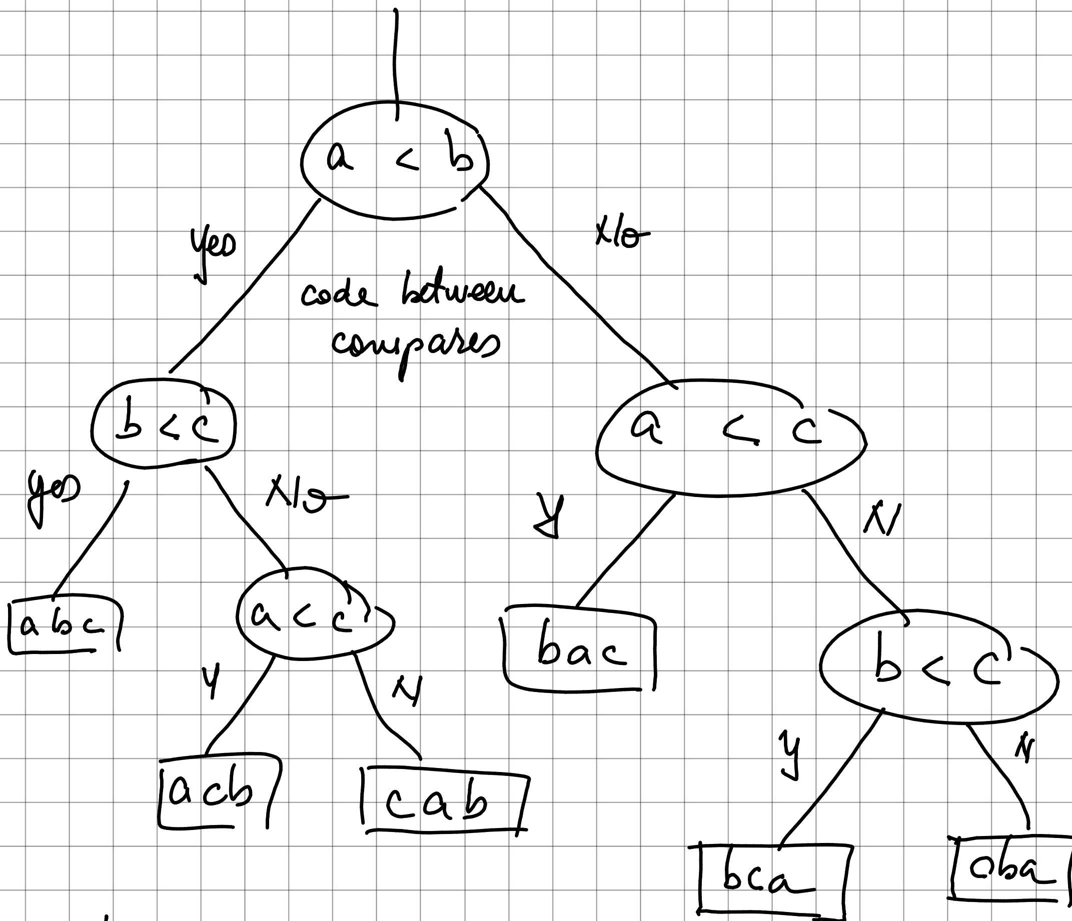
$\boxed{\text{Sorting}}$

- model of computation: decision tree

(can access info only through compares)

- cost model: # compares
- upper bound: ~ $N \lg N$ from mergesort
- lower bound?
- optimal?

a, b, c — <u>decision tree</u>



- <u>height of tree</u> → worst-case number of compares
- there has to be at least one leaf for each possible ordering

Proposition

Any compare based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst case

Proof

- assume array consists of N distinct values a_1 through a_N
- worst case is dictated by height of decision tree
- binary tree of height h can have at most 2^h leaves.
- $N!$ different orderings $\Rightarrow$ at least $N!$ leaves

$$\boxed{\begin{aligned} &2^h \geq \#\text{ leaves} \geq N! \\ &\Rightarrow h \geq \lg(N!) \sim N \lg N \end{aligned}}$$

Stirling's formula

=> lower bound ~ $N \lg N$

=> optimal sort ~ $N \lg N$

=> merge sort is an optimal algorithm

Mergesort

→ optimal with respect to # compares

→ not optimal with respect to space usage

Exceptions

- partially ordered arrays: we may not need $N \lg N$ compares

insertion sort requires $N-1$ compares if input array is sorted

- duplicate keys — input distribution of duplicates (3-way quicksort)
- digital properties of keys

Comparators

Comparator interface

- compare (key v, key w)
- must be a total order

* $\boxed{\text{PolarOrder Comparator (min. 6:23)}}$

for convex hull

Stability

- preserve previous ordering

- A stable sort preserves the relative order of items with equal keys -

* insertion sort and mergesort are stable

* selection sort and shell sort are not

$\boxed{\text{INSERTION SORT}}$ - stable

- equal items never move past each other

$\boxed{\text{SELECTION SORT}}$ – not stable

- long distance exchange might move an item past some equal item

$\boxed{\text{SHELL SORT}}$ – not stable

- long distance exchanges

$\boxed{\text{MERGE SORT}}$ – stable (as long as the merge operation is stable)