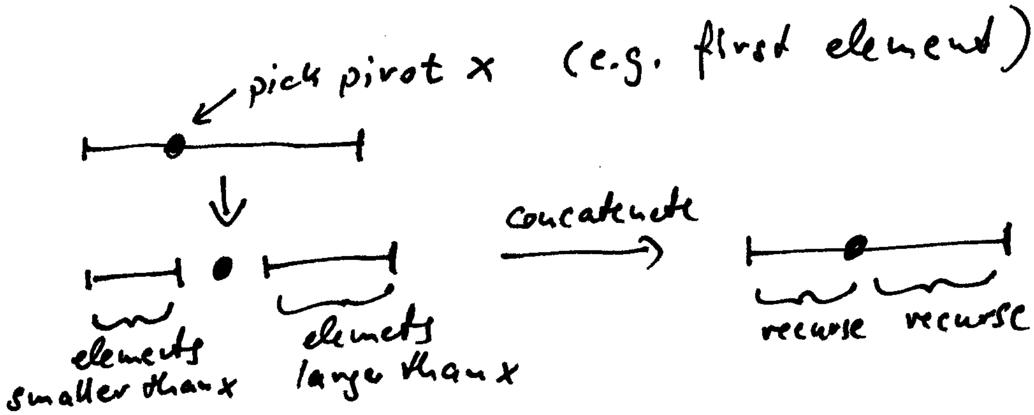
Sorting

We assume an array A of N integers to be sorted:
$A[1], \dots, A[N]$
visualized:

Quick sort

Basic Idea:



Better version: inplace
show for 3 7 8 5 2 1 9 5 4

Analysis:
best case: $O(N \log(N))$ (always pick a pivot that divides 50/50: $C(N) = C(N/2) + O(N)$)

worst case: $O(N^2)$ (always partition $1 + N-1$
(when) $C(N) = C(N-1) + O(N)$)

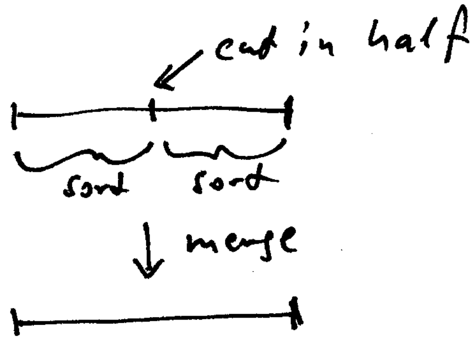
average case:
$\approx 1.39\, N \log_2(N) + O(N)$ comparisons
+ best among comparison-based algos
+ locality (spatial and temporal)
− worst case $O(N^2)$
− not good for small sizes (empirical)

Optimizations (Sedgwick 78)

- choose pivot = median (first, middle, last)
- use other sorting algos for small sizes
- choose multiple pivots (not worth it?)

Mergesort

Basic idea:



Show merging is $O(N)$

Analysis: $C(N) = 2C(N/2) + O(N)$

$\Rightarrow C(N) = O(N \log(N))$

+ locality (temporal and spatial)

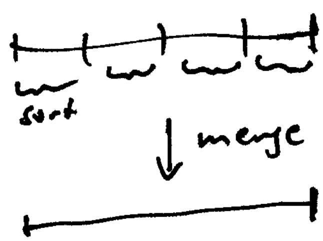
− $O(N)$ extra storage

Optimizations:

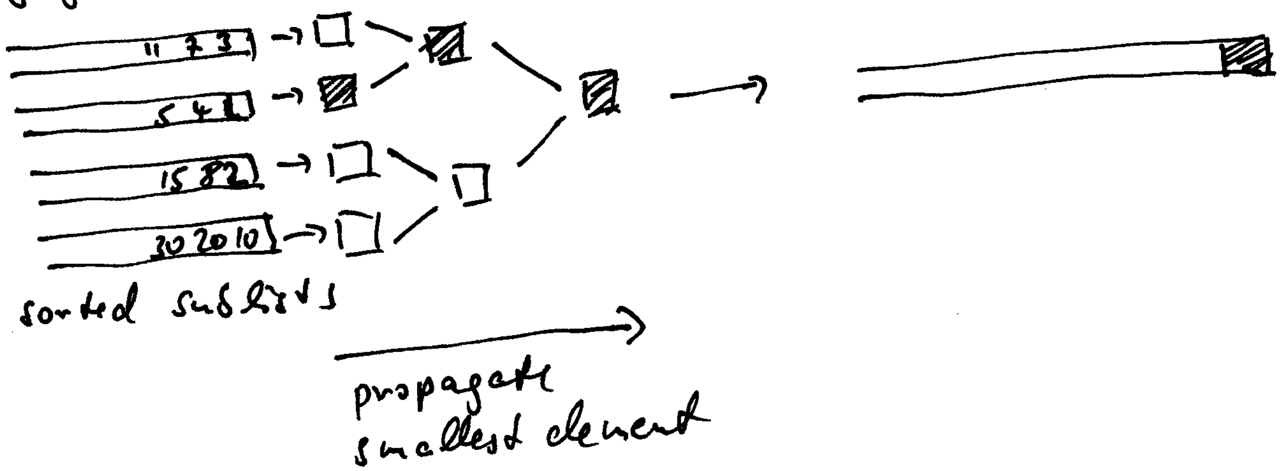
- other search for smaller problems
- divide into p chunks (N/p fits in cache, less overhead) → Multi-way mergesort

Multi-way Mergesort

Basic idea:



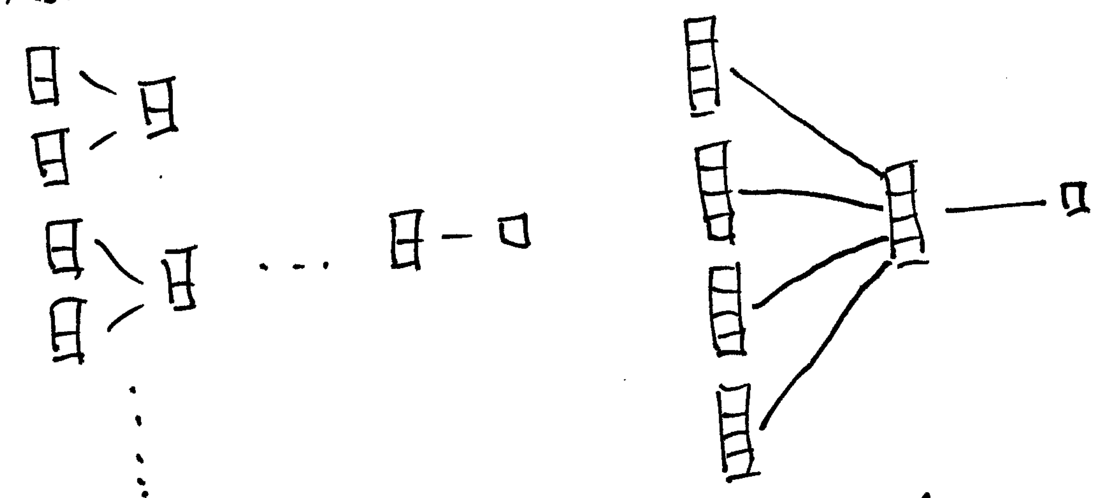
Merging:



Optimizations

- use p as degree of freedom
  (e.g., N/p and priority queue fit into cache)
- other search for smaller problems
- increase fan-out of priority queue to match cacheline size
  e.g. cacheline = 4 elements, then

instead of   do



Next, two algorithms suitable for small sizes

<u>Insertion Sort</u>

Basic idea: move through A and sort iteratively by swapping

Show code and an example

Analysis:

best: $O(N)$ (already sorted)

worst: $O(N^2)$ (reverse sorted)

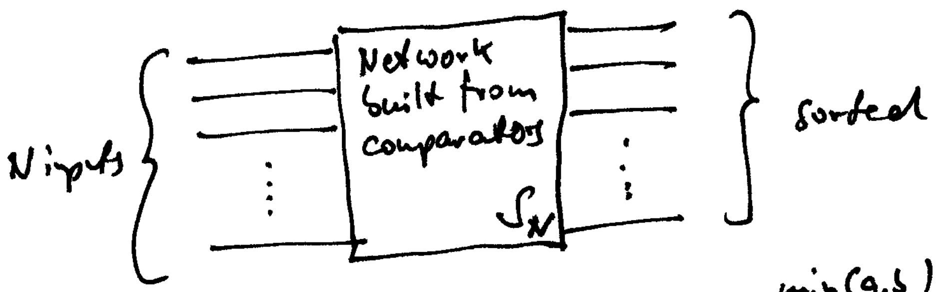
average: $O(N^2)$

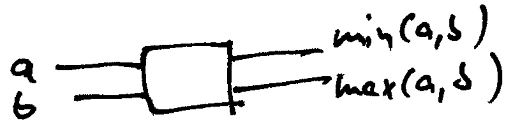
general: $O(N+d)$ $d = \{(i,j) \mid i<j \text{ and } A[i] > A[j]\}$

+ fast on almost sorted lists
- bad average cost
+ simple, inplace

Sorting Networks

Basic idea:

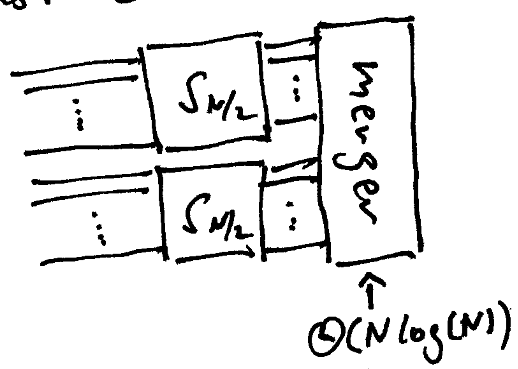


comparator: 

Show example

Constructions:

- best known uses $O(N \log N)$ comparators but huge constant
- useful constructions have $O(N \log^2 N)$ comparators construction: (recursive)



$$C(N) = 2C(N/2) + O(N \log N)$$
$$\Rightarrow C(N) = O(N \log^2 N)$$

Analysis:

+ independent of input data, inplace
+ $O(N \log^2 N)$ worst case
- " best case

Optimizations:

- schedule comparisons, e.g., for instruction level parallelism