universität freiburg

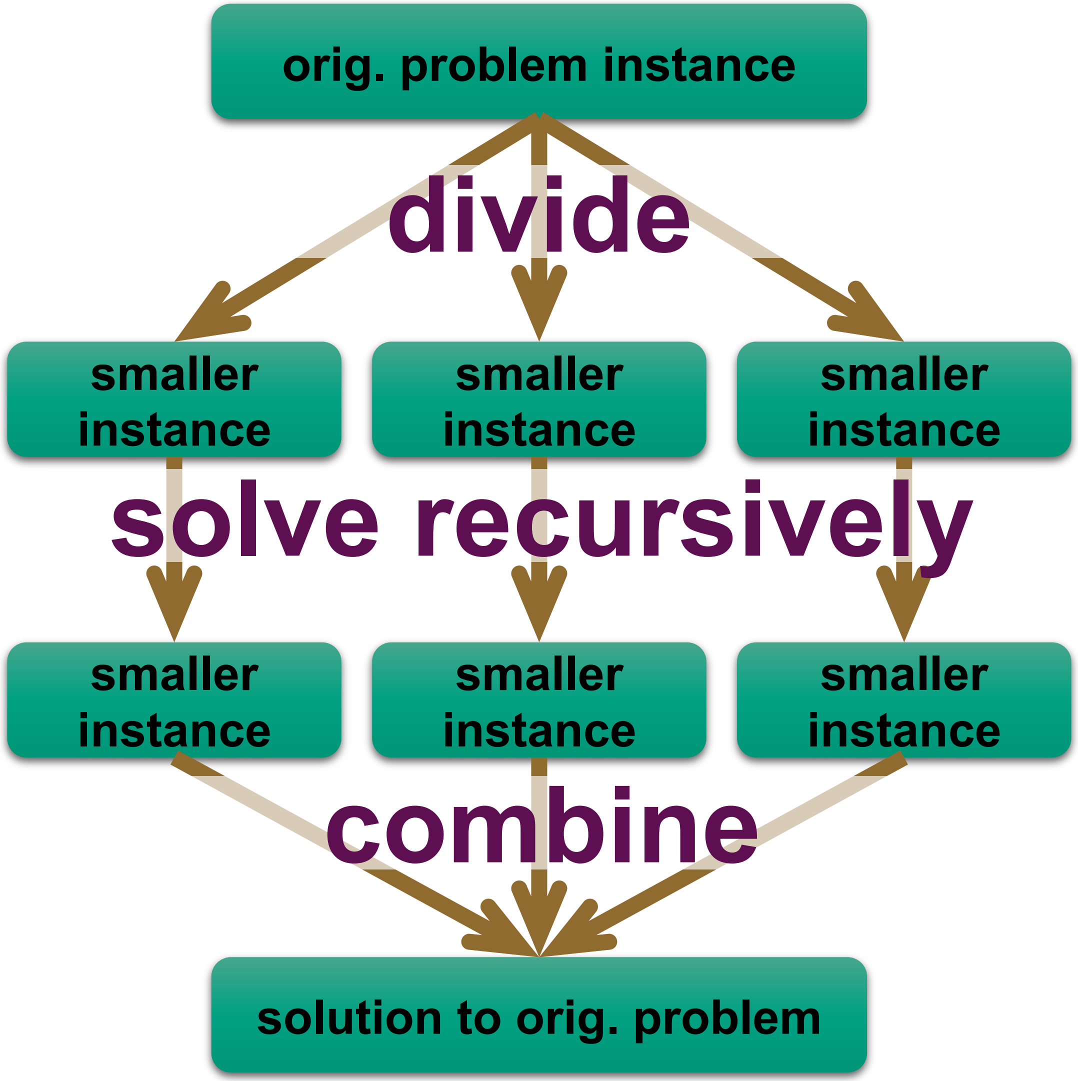
Algorithm Theory – WS 2024/25

Chapter 1 : Divide and Conquer Algorithms – Part I

Fabian Kuhn
Dept. of Computer Science
Algorithms and Complexity

Divide and Conquer Algorithms: Overview

- Important and very powerful algorithm design method
- **Highlevel idea:**
  1) **Divide** a given problem instance **into** several **smaller instances** of the same kind
  2) **Solve** the smaller instances **recursively**
  3) **Combine/merge** solutions of small instances to obtain a solution for the original problem



Divide and Conquer Algorithms: Examples

Examples from your basic algorithms and data structures lecture

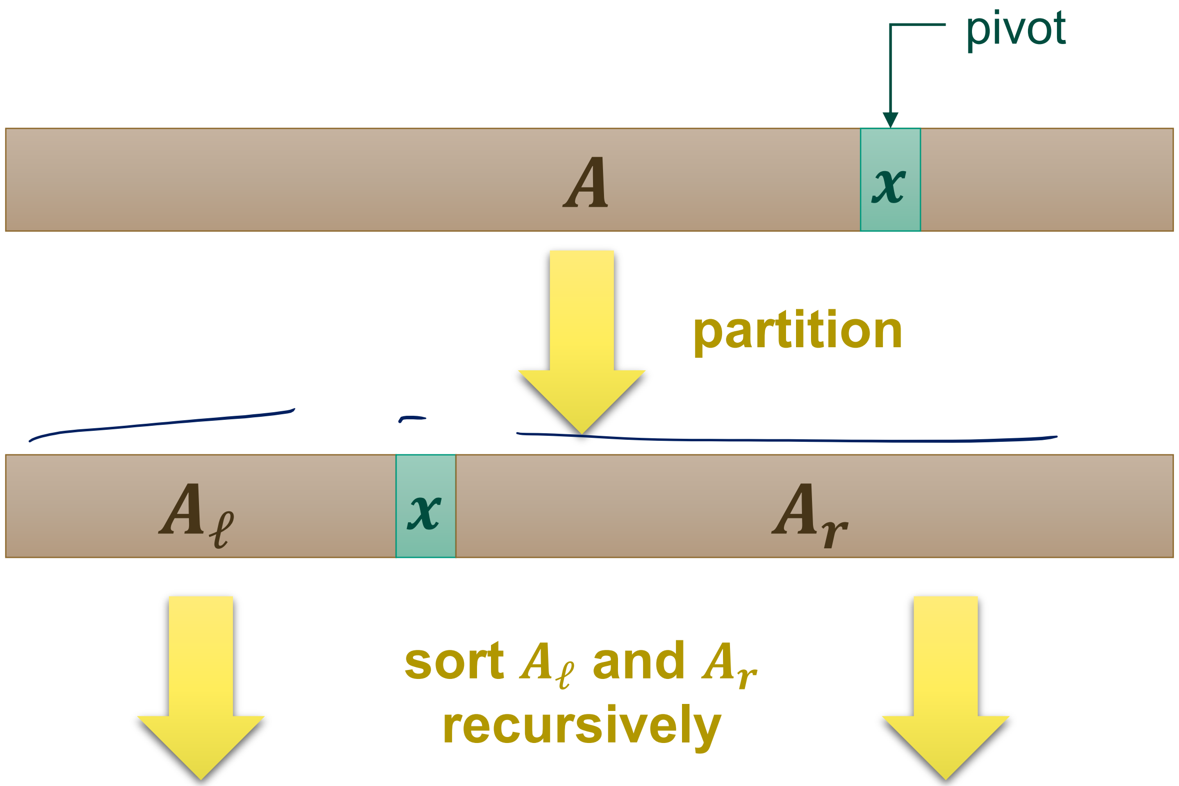
- Sorting: Mergesort and Quicksort
- Searching: Binary Search

Other examples

- Computing the median value
- Computing the difference between two global orders
- Geometry problems: convex hull, Delaunay triangulation, Voronoi diagram, line intersection
  closest pair of points
- Polynomial / integer multiplication, Fast Fourier Transform (FFT) algorithm
- …

Divide and Conquer Example: Quicksort

Goal: sort array A

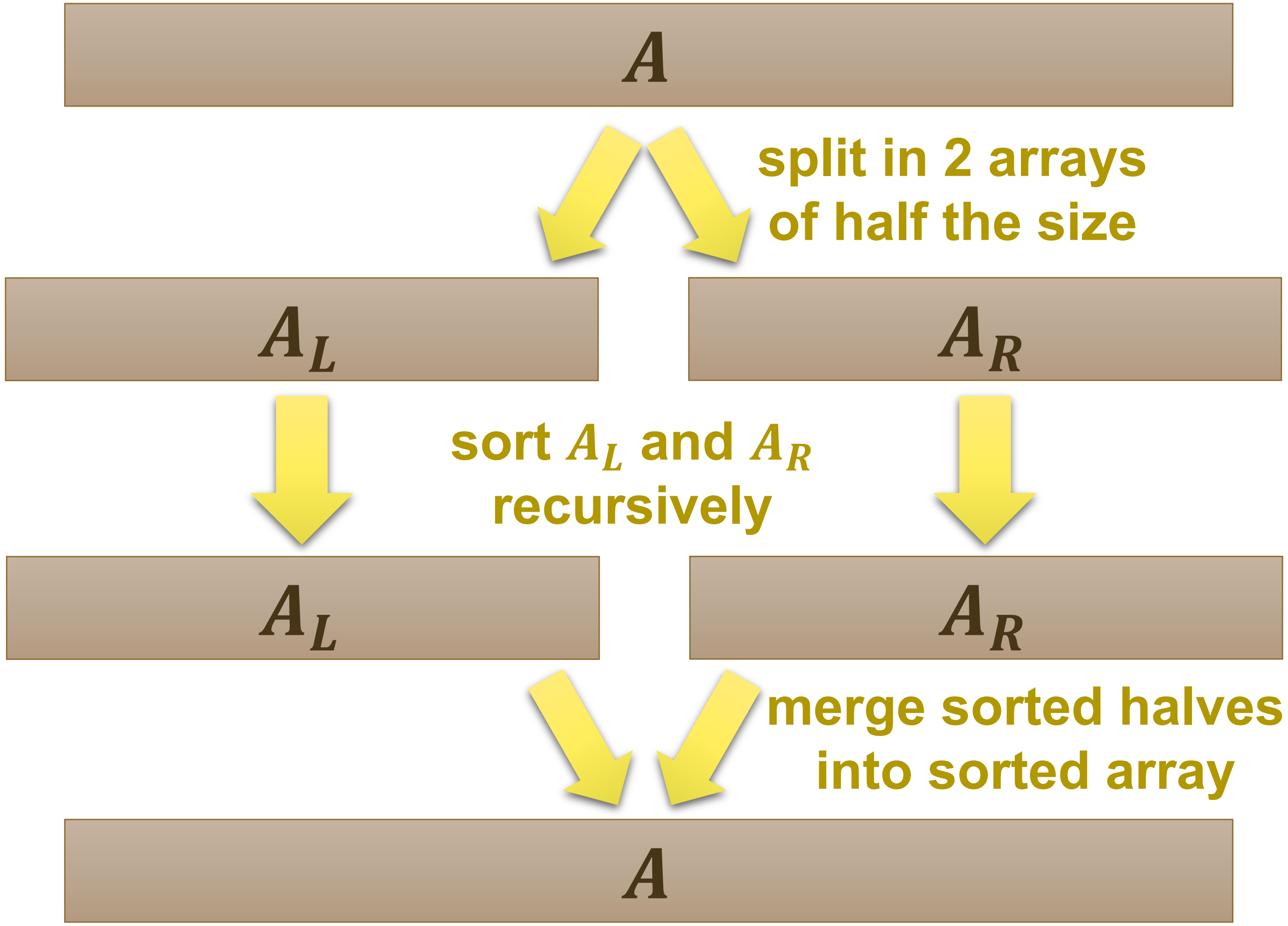


Pseudocode

```
function QuickSort(A):
    if size(A) > 1 then
        choose pivot element x in A
        partition A into
          A_ℓ with elements ≤ x and
          A_r with elements ≥ x
        QuickSort(A_ℓ)        // sort A_ℓ recursively
        QuickSort(A_r)        // sort A_r recursively
```

Divide and Conquer Example: Mergesort

Goal: sort array A



Pseudocode

$A[l .. r-1]$

```
function MergeSort(A, l, r):
    if r > l then
        m := (r + l) / 2      // integer division
        MergeSort(A, l, m)    // sort left half rec.
        MergeSort(A, m, r)    // sort right half rec.
        merge(A, l, m, r)
                // merge sorted left and right halves
                // A[l..m-1] and A[m..r-1] into sorted
                // array A[l..r-1]
```

Divide and Conquer: Highlevel Principle

Divide-and-conquer method for solving a problem instance of size n:

1. Divide

$n \leq c$: Solve the problem directly.

$n > c$: Divide the problem into k subproblems of sizes $n_1, \ldots, n_k < n$ $(k \geq 2)$. (k=1 possible)

2. Conquer

Solve the k subproblems in the same way (typically by using recursion).

3. Combine

Combine the partial solutions to generate a solution for the original instance.

QS	MS
choose pivot & partition	divide in middle
recursion	recursion
-	merge sorted halves

Divide and Conquer: Analysis

Recurrence relation:

- $\boldsymbol{T(n)}$: max. number of steps for solving an instance of size n
- $\boldsymbol{T(n)} = \begin{cases} c & \text{if } n \leq n_0 \\ T(n_1) + \cdots + T(n_k) & \text{if } n > n_0 \\ \quad + \text{cost for divide and combine} & \end{cases}$

Important Special Case: $\boldsymbol{k = 2, n_1 = n_2 = {}^{n}/_{2}}$

- Cost for divide and combine: $\mathrm{DC}(n)$
- $T(1) = c$
- $T(n) = 2 \cdot T(n/2) + \mathrm{DC}(n)$

Mergesort: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \implies T(n) = O(n \cdot \log n)$

Cost for Divide and Combine:

Quicksort:

- Divide (find pivot & partition): $O(n)$
- Combine (-): $O(1)$

Mergesort:

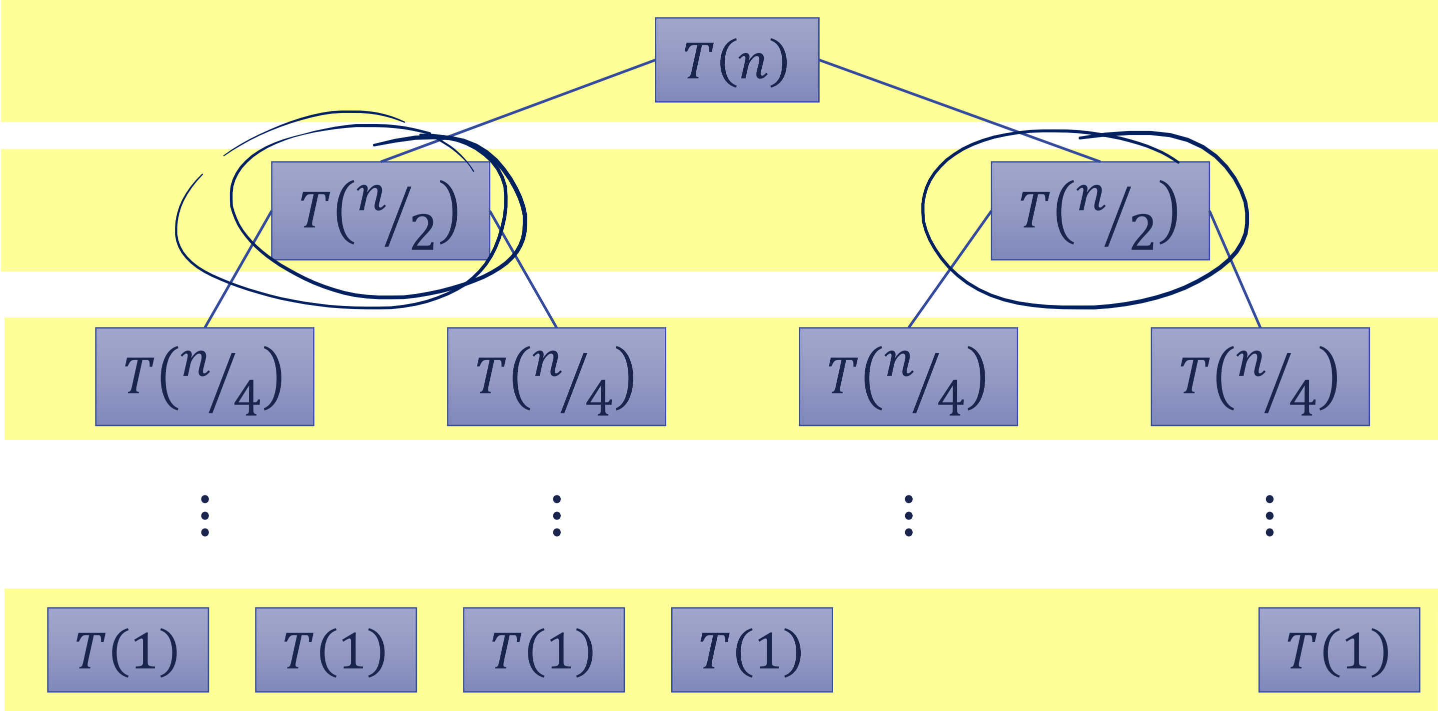
- Divide (split in middle): $O(1)$
- Combine (merge halves): $O(n)$

Analysis Example: Mergesort

Recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + cn, \qquad T(1) \leq c$$

Guess the solution by drawing the recursion tree :



$$cn$$

$$2 \cdot c \cdot \frac{n}{2} = cn$$

$$4 \cdot c \cdot \frac{n}{4} = cn$$

$$n \cdot T(1) \leq cn$$

Total time:

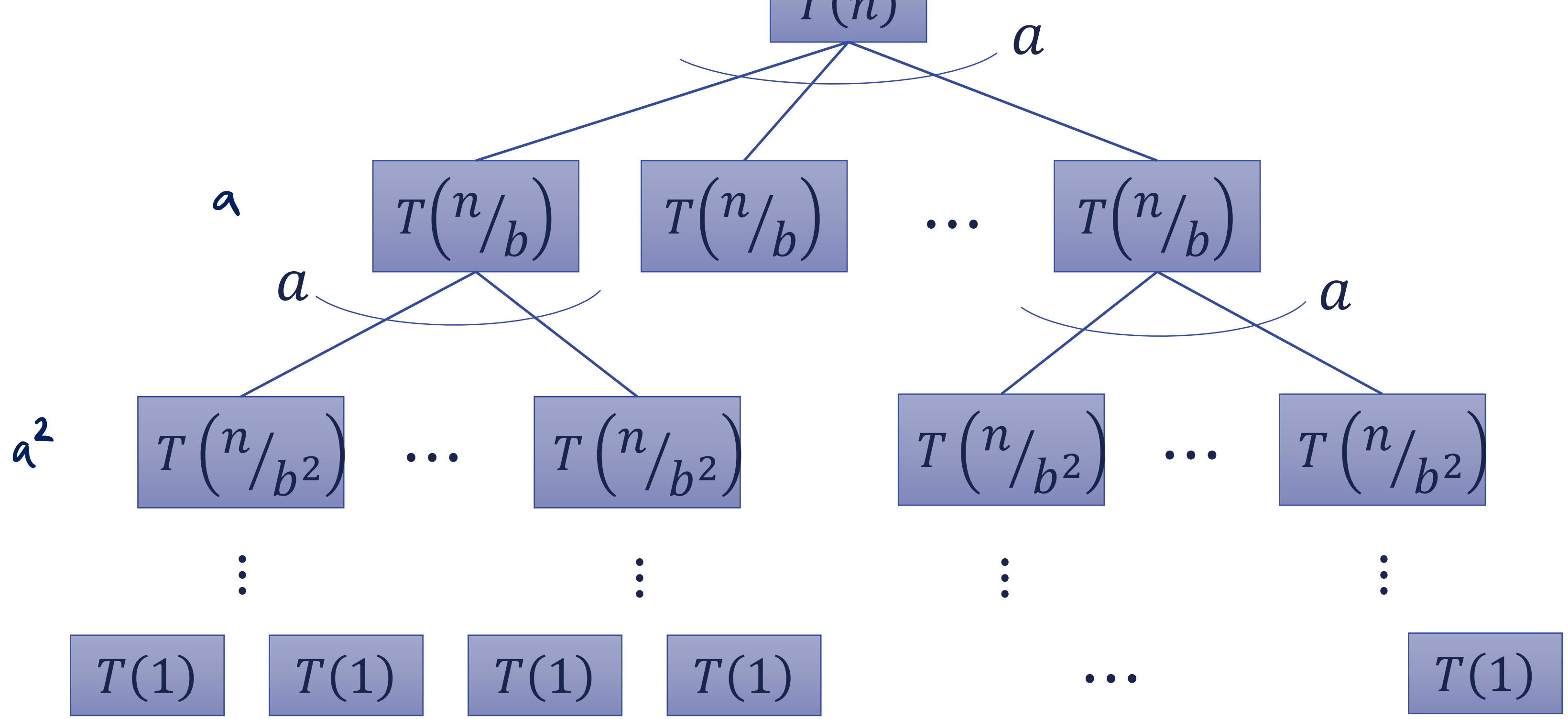
$$(1 + \log_2 n) \cdot cn$$

More General Recurrence Relations

Recurrence relation:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^c), \qquad T(n) = O(1) \text{ for } n \leq n_0$$

Obtain Intuition by Looking at Recursion:



More General Recurrence Relations

$b^k = n$

$\left(\frac{a}{b^c}\right)^\ell \cdot n^c$

Recurrence relation:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^c), \qquad T(n) = O(1) \text{ for } n \le n_0$$

$\frac{a}{b^c} < 1 \qquad \frac{a}{b^c} > 1$

Obtain Intuition by Looking at Recursion:

Rec. Level	Subproblem Size	#Subproblems	Time
1	n	1	$1 \cdot n^c$ ←
2	n/b	a	$a \cdot (n/b)^c = \frac{a}{b^c} \cdot n^c$
3	n/b^2	a^2	$a^2 \cdot (n/b^2)^c = \left(\frac{a}{b^c}\right)^2 \cdot n^c$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_b n$	1	$a^{\log_b n}$	$a^{\log_b n} \cdot 1 = n^{\log_b a}$

More General Recurrence Relations

$f\left(\frac{n}{b}\right) \leq \frac{f(n)}{b^c}$

$f\left(\frac{n}{b}\right) \geq \alpha \cdot \left(\frac{n}{b}\right)^c$

Recurrence relation:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^c), \qquad T(n) = O(1) \text{ for } n \leq n_0$$

Obtain Intuition by Looking at Recursion:

Observations:

- Time grows/shrinks by factor (a/b^c) per level
- If $a/b^c < 1$ $(c > \log_b a)$, first level dominates:
  $T(n) = O(n^c)$
- If $a/b^c > 1$ $(c < \log_b a)$, last level dominates:
  $T(n) = O\left(n^{\log_b a}\right)$
- If $a/b^c = 1$ $(c = \log_b a)$, all levels are the same:
  $T(n) = O(n^c \cdot \log n)$

$a \cdot f\left(\frac{n}{b}\right)$ $\qquad 1 \cdot n^c$
$a \cdot (n/b)^c = \frac{a}{b^c} \cdot n^c$
$a^2 \cdot (n/b^2)^c = \left(\frac{a}{b^c}\right)^2 \cdot n^c$
$\vdots$
$a^{\log_b n} \cdot 1 = n^{\log_b a}$

Recurrence Relations: Master Theorem

Recurrence relation:

$$T(n) = \underline{a \cdot T\left(\frac{n}{b}\right)} + \underline{f(n)}, \qquad T(n) = O(1) \text{ for } n \leq n_0$$

Cases:

- $f(n) = O(n^c),\ c < \log_b a$

  $\Theta(n^c)$?

  $$\boldsymbol{T(n) = \Theta\left(n^{\log_b a}\right)}$$

- $f(n) = \Omega(n^c),\ c > \log_b a$

  $$\boldsymbol{T(n) = \Theta\big(f(n)\big)}$$

- $f(n) = \Theta\left(n^c \cdot \log^k n\right),\ c = \log_b a$

  $$\boldsymbol{T(n) = \Theta\left(n^c \cdot \log^{k+1} n\right)}$$

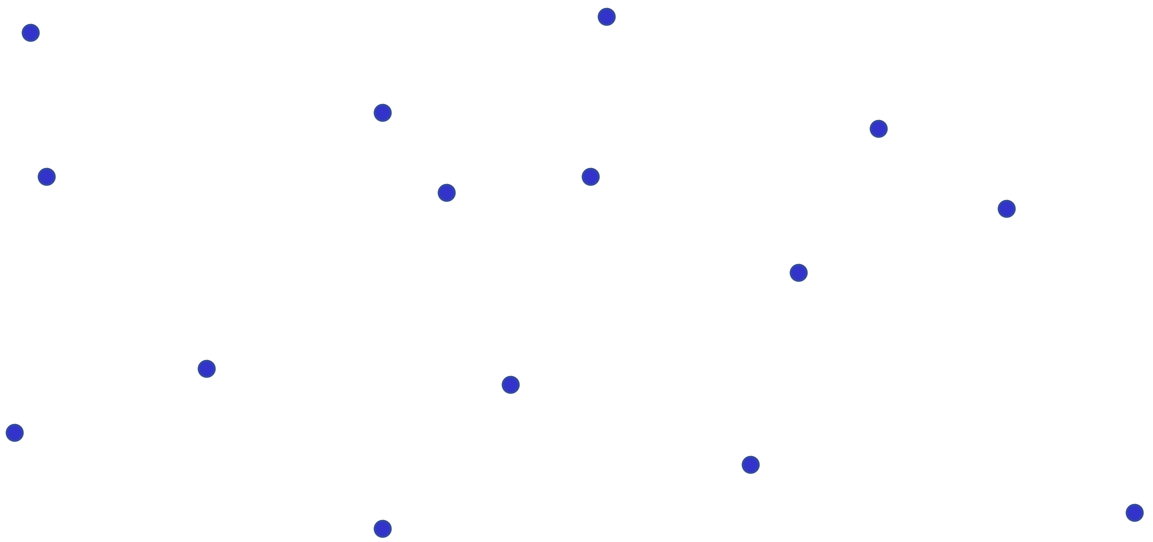
$$\log^{(k)} x = \overbrace{\log \log \ldots \log}^{k \text{ times}} (x)$$

$$\log^k x = (\log x)^k$$

Geometric divide-and-conquer

Closest Pair Problem: in $\mathbb{R}^2$

- Given a set S of n points, find a pair of points with the smallest distance.



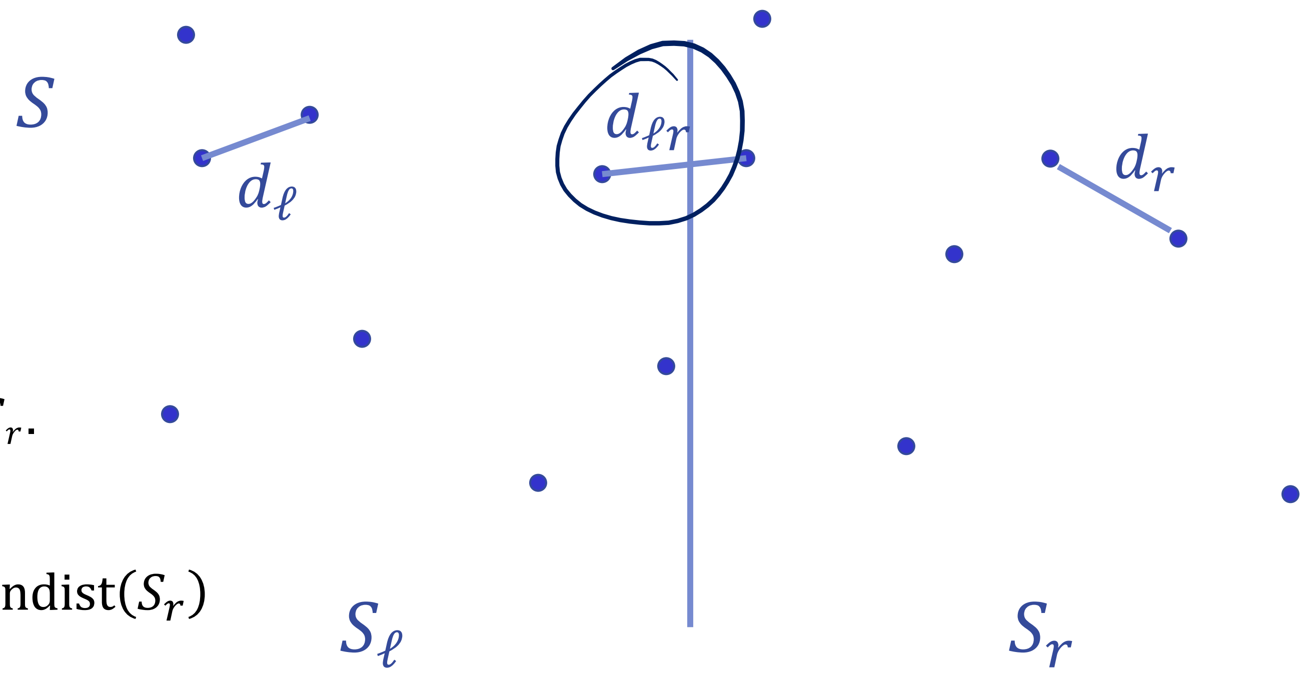
Naïve solution:

- Go over all pairs of points, compute distance, take minimum
- Time: $O(n^2)$

Divide-and-Conquer Solution

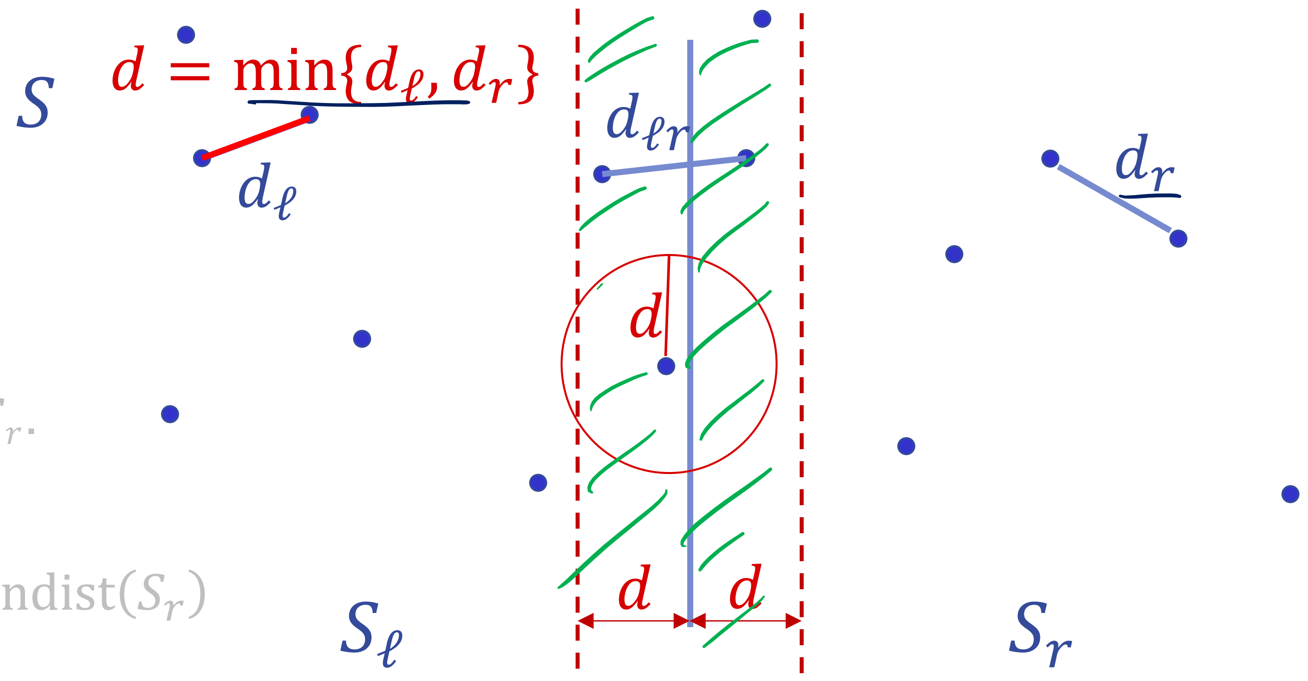
1. Sort points by x-coordinate
2. Divide:
   - Divide S into two equal sized sets S_ℓ und S_r.
3. Conquer:
   - Recursively find $d_\ell = \mathrm{mindist}(S_\ell),\ d_r = \mathrm{mindist}(S_r)$
4. Combine:
   - Define $d \coloneqq \min\{d_\ell, d_r\}$
   - Compute $d_{\ell r} \coloneqq \min\{d(a,b) : a \in S_\ell, b \in S_r\}$

only needed
if $d_{\ell r} < \min\{d_\ell, d_r\}$

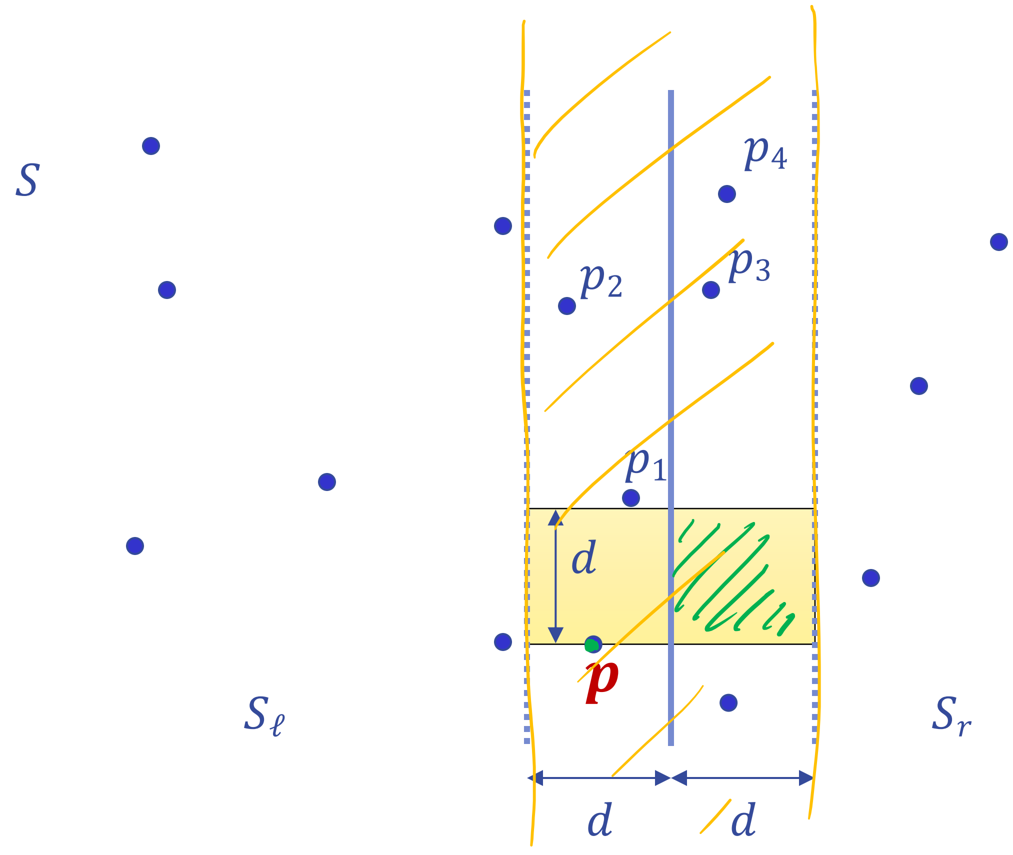


Divide-and-Conquer Solution

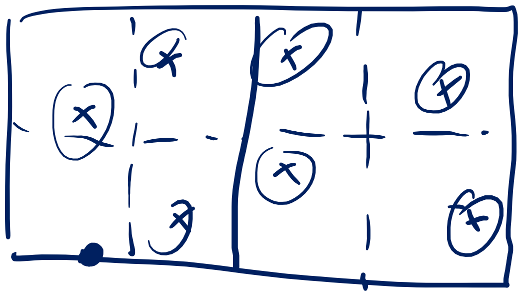
1. Sort points by x-coordinate
2. Divide:
   - Divide S into two equal sized sets S_ℓ und S_r.
3. Conquer:
   - Recursively find $d_\ell = \text{mindist}(S_\ell),\ d_r = \text{mindist}(S_r)$
4. **Computation of $d_{\ell r}$ if $d_{\ell r} < d$:**
   - Points $a \in S_\ell$ and $b \in S_r$ must be within distance d of the dividing line between S_ℓ and S_r

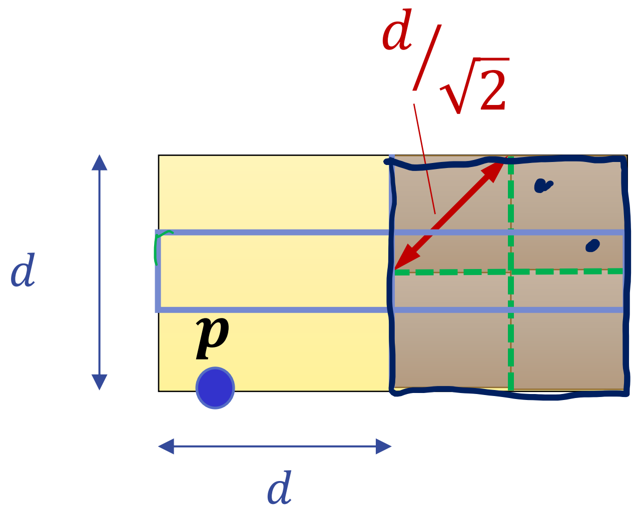


Combine step



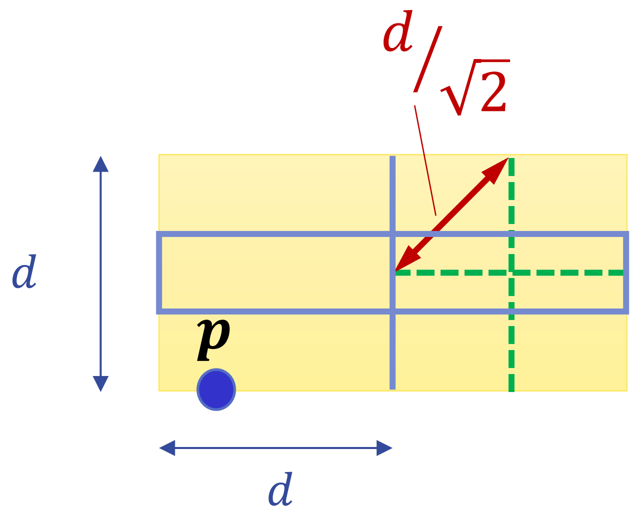
$d = \min\{d_\ell, dr\}$





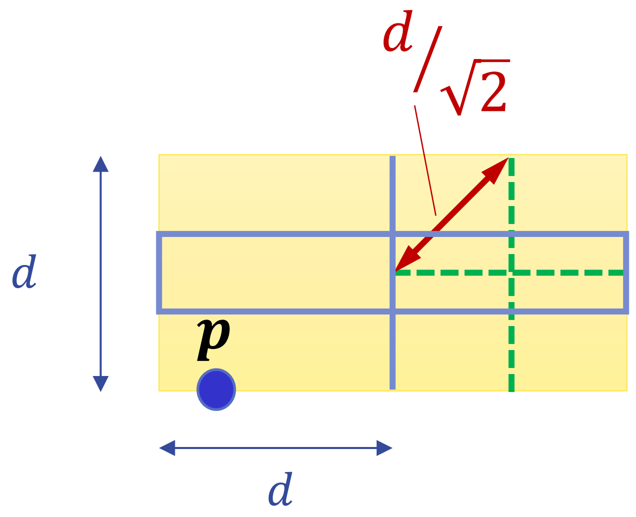
Divide-and-Conquer Solution

1. Consider only points within distance $\leq d$ of the bisection line, in the order of increasing y-coordinates.
2. For each point p consider all points q on the other side which are within y-distance less than d
   - It suffices to consider the points q with equal or larger y-coordinate
3. There are at most 4 such points!



Divide-and-Conquer Solution

- Initially sort the points in S in order of increasing x-coordinates
- While computing closest pair, sort S according to y-coordinates
  - Partition S into S_ℓ and S_r, solve and sort sub-problems recursively
    - Thus, when combining S_ℓ and S_r, points in each part are sorted by y-coordinates
  - Merge to get sorted S according to y-coordinates
  - Center points: points within x-distance $d = \min\{d_\ell, d_r\}$ of center
  - Go through center points in S in order of incr. y-coordinates
    - Each point only has to be compared to the 7 next center points in the sorted order of all center points (when including the center points on the same side)



Running Time

Recurrence relation:

$$T(n) = \underline{2 \cdot T(n/2)} + c \cdot n, \qquad T(1) \leq c$$

Solution:

- Same as for computing number of Mergesort (and many others…)

$$T(n) = O(n \cdot \log n)$$