Chapter 1
Divide and Conquer

Algorithm Theory
WS 2015/16

Fabian Kuhn

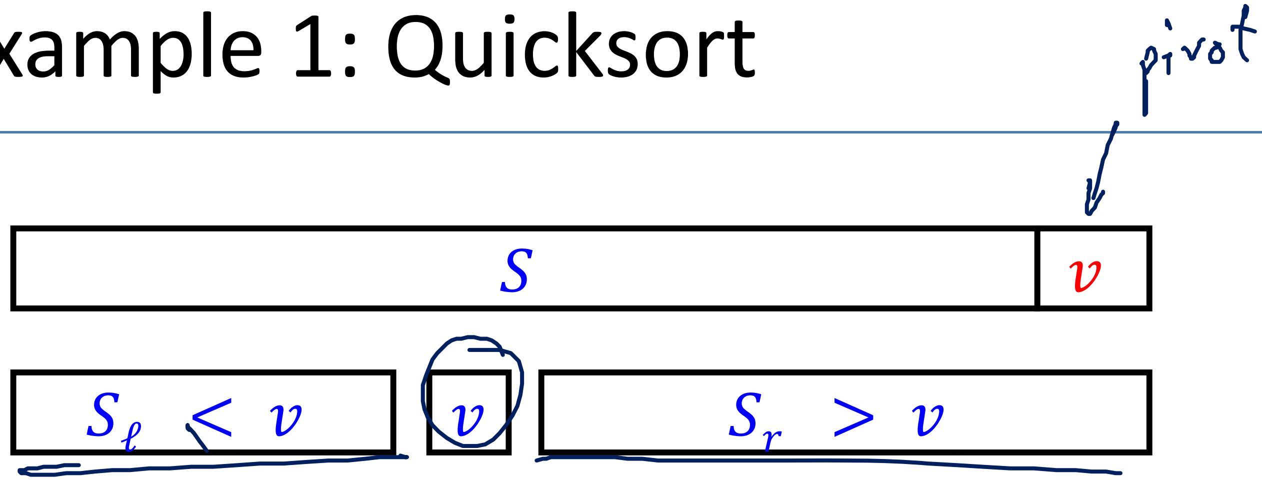
Divide-And-Conquer Principle

- Important algorithm design method

- Examples from Informatik 2:
  - Sorting: Mergesort, Quicksort
  - Binary search can be considered as a divide and conquer algorithm

- Further examples
  - Median
  - Compairing orders
  - Delaunay triangulation / Voronoi diagram
  - Closest pairs
  - Line intersections
  - Polynomial multiplication / FFT
  - ...

Example 1: Quicksort



function Quick (S: sequence): sequence;

{returns the sorted sequence S}

begin

  if $\#S \leq 1$ then **return** S

  else { choose pivot element v in S;

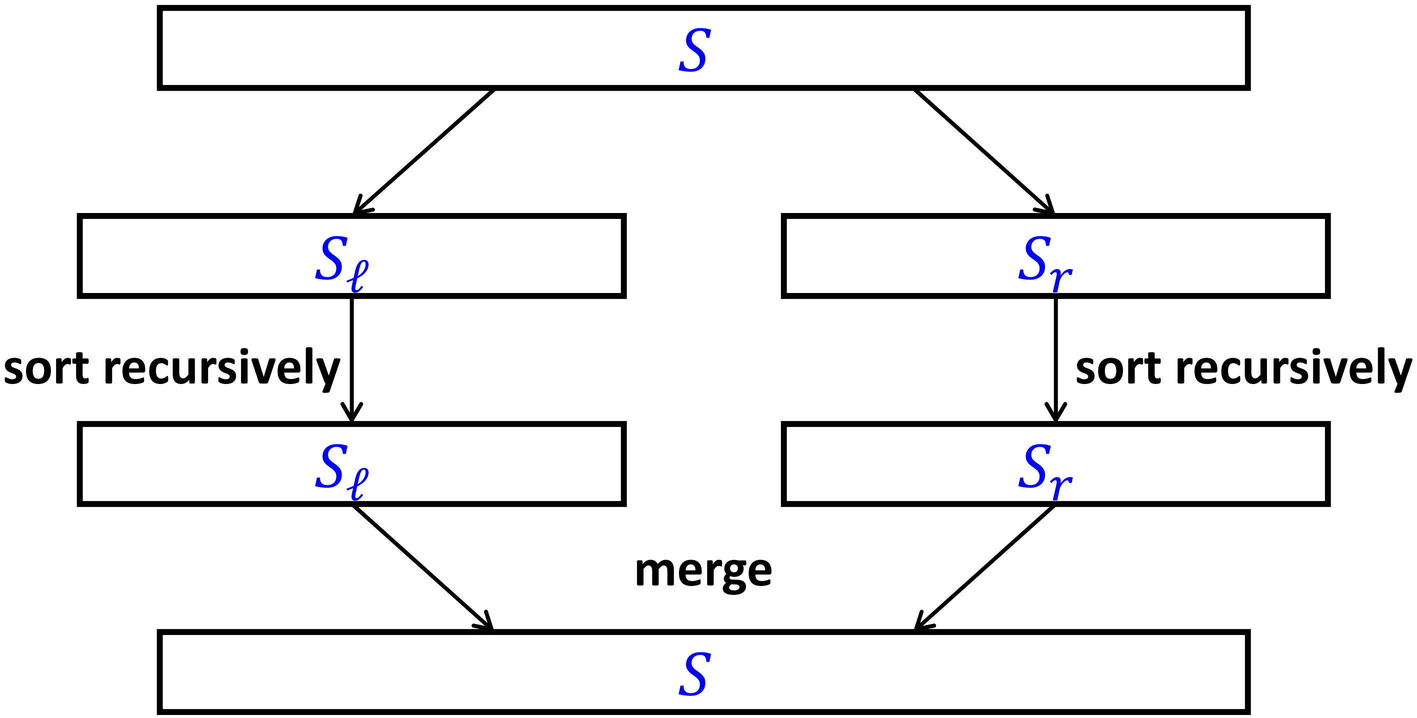
    partition S into S_ℓ with elements $< v$,

    and S_r with elements $> v$

    return $\boxed{\text{Quick}(S_\ell) \mid v \mid \text{Quick}(S_r)}$

end;

Example 2: Mergesort



Formulation of the D&C principle

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

1. Divide

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

$n > c$: Divide the problem into k subproblems of sizes $n_1, \dots, n_k < n$ $(k \ge 2)$.

2. Conquer

Solve the k subproblems in the same way (recursively).

3. Combine

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

QS

partition

sort rec.

sort rec

merge

Analysis

Recurrence relation:

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

$\lfloor \frac{n}{2} \rfloor$ $\lceil \frac{n}{2} \rceil$

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

- cost for divide and combine: $\mathrm{DC}(n)$
- $T(1) \leq a$
- $T(n) \leq 2T(n/2) + \mathrm{DC}(n)$

mergesort: $T(n) = 2 \cdot T(n/2) + O(n)$

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

Analysis, Example

Recurrence relation:

$$\underline{T(n)} \leq \underline{2} \cdot \underline{T(n/2)} + \underline{cn}^2, \qquad \underline{T(1) \leq a}$$

Guess the solution by repeated substitution:

$$
\begin{aligned}
T(n) &\leq 2 \cdot T(n/2) + cn^2 \\
&\leq 2\left(2 \cdot T(n/4) + c \cdot \left(\tfrac{n}{2}\right)^2\right) + cn^2 \\
&= 4 \cdot T(n/4) + \left(1 + \tfrac{1}{2}\right) c \cdot n^2 \\
&\leq 4\left(2 \cdot T(n/8) + c\left(\tfrac{n}{4}\right)^2\right) + \left(1 + \tfrac{1}{2}\right) cn^2 \\
&= 8T(n/8) + \left(1 + \tfrac{1}{2} + \tfrac{1}{4}\right) cn^2 \\
&\;\;\vdots \\
&\leq 2^i\, T(\underbrace{n/2^i}_{1}) + \left(1 + \tfrac{1}{2} + \dots + \tfrac{1}{2^{i-1}}\right) cn^2 \\
&\;\;\vdots \\
&< n \cdot \underbrace{T(1)}_{a} + 2cn^2 \leq \underline{\underline{a \cdot n + 2cn^2}}
\end{aligned}
$$

Analysis, Example

Recurrence relation:

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

Verify by induction:

Guess: $T(n) \leq a \cdot n + 2 \cdot c \cdot n^2$

Induction:

Base: $T(1) \leq a + 2c$ ✓

Step: $T(n) \leq 2T(n/2) + cn^2$

$$\overset{\text{(I.H.)}}{\leq} 2 \cdot \left(a\frac{n}{2} + 2c \cdot \frac{n^2}{4}\right) + cn^2$$

$$= a \cdot n + 2cn^2$$

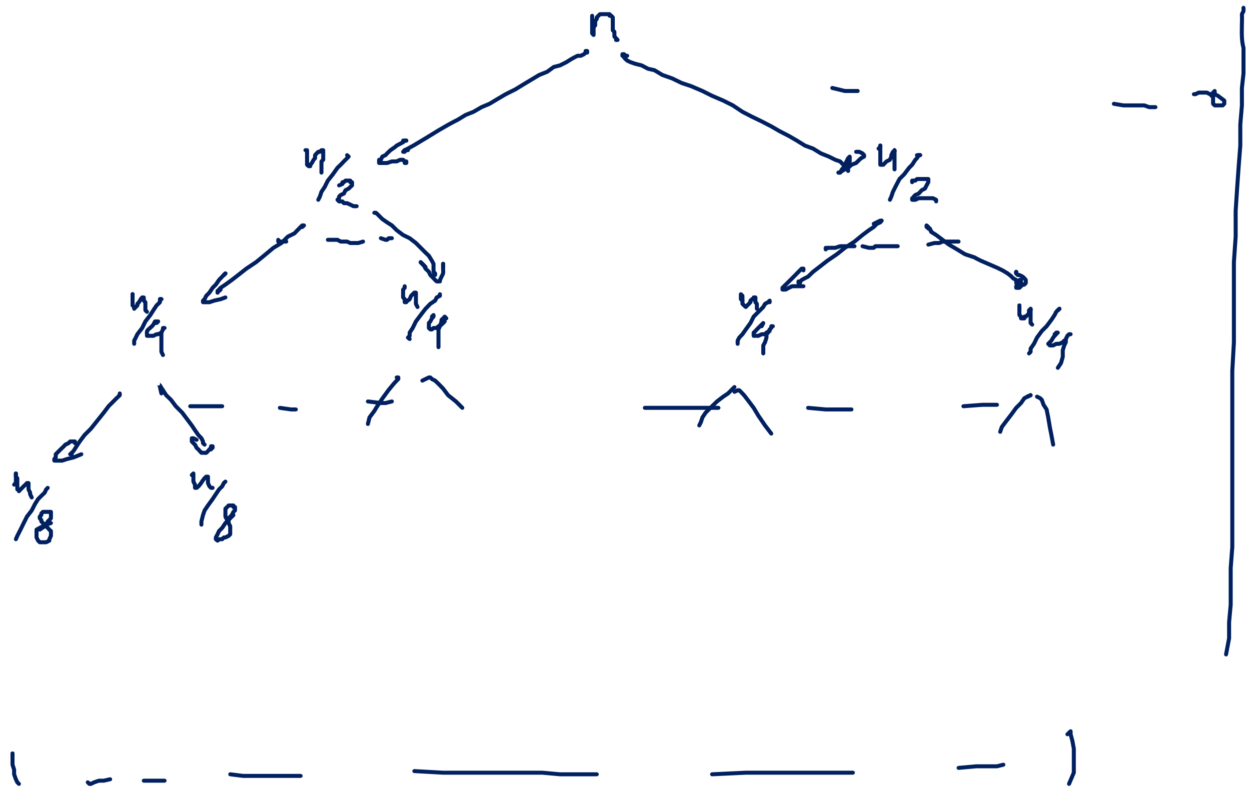
$\square$

Analysis, Example

Recurrence relation:

$$T(n) \le 2 \cdot T(n/2) + cn^2, \qquad T(1) \le \underline{a}$$

Guess the solution by drawing the recursion tree:



divide & combine

$c \cdot n^2$

$2 \cdot c \cdot \left(\frac{n}{2}\right)^2 = \frac{c}{2} \cdot n^2$

$4 \cdot c \left(\frac{n}{4}\right)^2 = \frac{c}{4} n^2$

$\vdots$

$n \cdot a$

Comparing Orders

- Many web systems maintain user preferences / rankings on things like books, movies, restaurants, …

- Collaborative filtering:
  - Predict user taste by comparing rankings of different users.
  - If the system finds users with similar tastes, it can make recommendations (e.g., Amazon)

- Core issue: Compare two rankings
  - Intuitively, two rankings (of movies) are more similar, the more pairs are ordered in the same way
  - Label the first user's movies from 1 to n according to ranking
  - Order labels according to second user's ranking
  - How far is this from the ascending order (of the first user)?

Number of Inversions

Formal problem:

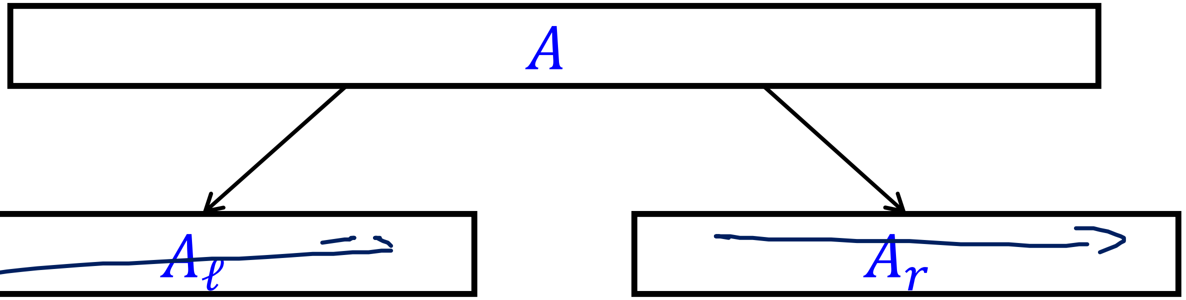
- **Given**: array $A = [a_1, a_2, a_3, \dots, a_n]$ of distinct elements
- **Objective**: Compute number of inversions I

$$I := \left|\{0 \leq i < j \leq n \mid a_i > a_j)\}\right|$$

- **Example**: $A = [\ 4\ ,\ 1\ ,\ 5\ ,\ 2\ ,\ 7\ ,\ 10\ ,\ 6\ ]$ 5 inversions
- **Naive solution**: go through all pairs

time: $O(n^2)$

Divide and conquer



1. Divide array into 2 equal parts A_ℓ and A_r
2. Recursively compute #inversions in A_ℓ and A_r
3. Combine: add #pairs $a_i \in A_\ell$, $a_j \in A_r$ such that $a_i > a_j$

Combine Step: Example

7 7 6

- Assume A_ℓ and A_r are sorted

3	5	8	13	14	18	24	25	30

6	7	9	19	21	23	28	32	33

i i i i i i i i i

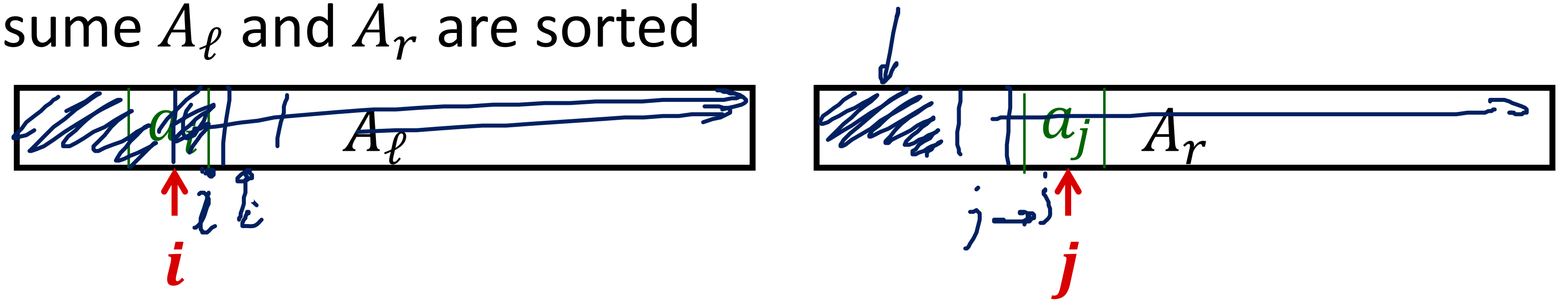
j j j j j j j j j

$IC = 7 + 7 + 6 + 3 + 3 + 3 + 1$

3	5	6															

Combine Step

Assume A_ℓ and A_r are sorted



Idea:

- Maintain pointers i and j to go through the sorted parts
- While going through the sorted parts, we merge the two parts into one sorted part (like in MergeSort)

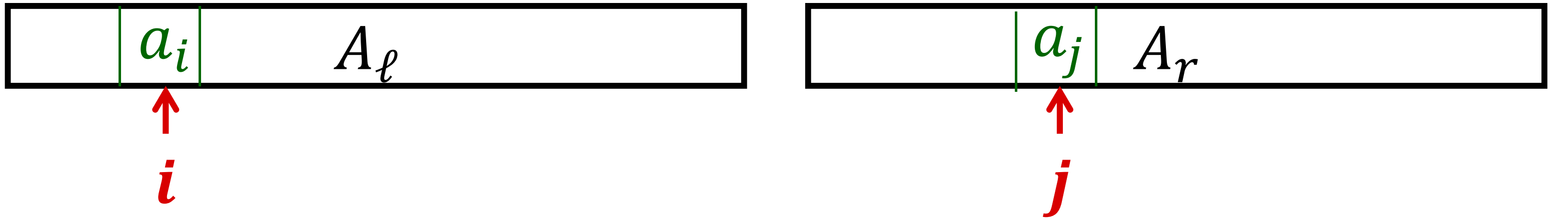
  and we count the number of inversions between the parts

Invariant:

- At each point in time, all inversions involving some element left of i (in A_ℓ) or left of j (in A_r) are counted
  - and all others still have to be counted...

Combine Step

Assume A_ℓ and A_r are sorted



- Pointers i and j, initially pointing to first elements of A_ℓ and A_r
- If $a_i \leq a_j$:
  - a_i is smallest among the remaining elements
  - No inversion of a_i and one of the remaining elements
  - Do not change count
- If $a_i > a_j$:
  - a_j is smallest among the remaining elements
  - a_j is smaller than all remaining elements in A_ℓ
  - Add number of remaining elements in A_ℓ to count
- Increment point, pointing to smaller element

Combine Step

- Need sub-sequences in sorted order
- Then, combine step is like merging in merge sort

- **Idea**: Solve sorting and #inversions at the same time!
  1. Partition A into two equal parts A_ℓ and A_r
  2. Recursively compute #inversions and sort A_ℓ and A_r

     $2 \cdot T(n/2)$

  3. Merge A_ℓ and A_r to sorted sequence, at the same time, compute number of inversions between elements a_i in A_ℓ and a_j in A_r

     linear in n

Analysis, Guessing

Recurrence relation:

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

Repeated substitution:

$$\begin{aligned}
T(n) &\leq 2\,T(n/2) + cn \\
&\leq 4 \cdot T(n/4) + 2cn \\
&\leq 8 \cdot T(n/8) + 3cn \\
&\;\;\vdots \\
&\leq 2^i\,T(n/2^i) + i \cdot c \cdot n \\
&\;\;\vdots \\
&\leq n \cdot T(1) + c \cdot n \cdot \log_2 n \leq c \cdot n(1 + \log_2 n)
\end{aligned}$$

Analysis, Induction

Recurrence relation:

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

Verify by induction:

$T(n) \leq cn(1 + \log n)$

$\underline{\text{Base}}$: $\underline{n=1}$: $T(1) \leq c \cdot 1 \, (1 + 0) = c$ ✓

$\underline{\text{Step}}$: $T(n) \leq 2T(n/2) + c \cdot n$

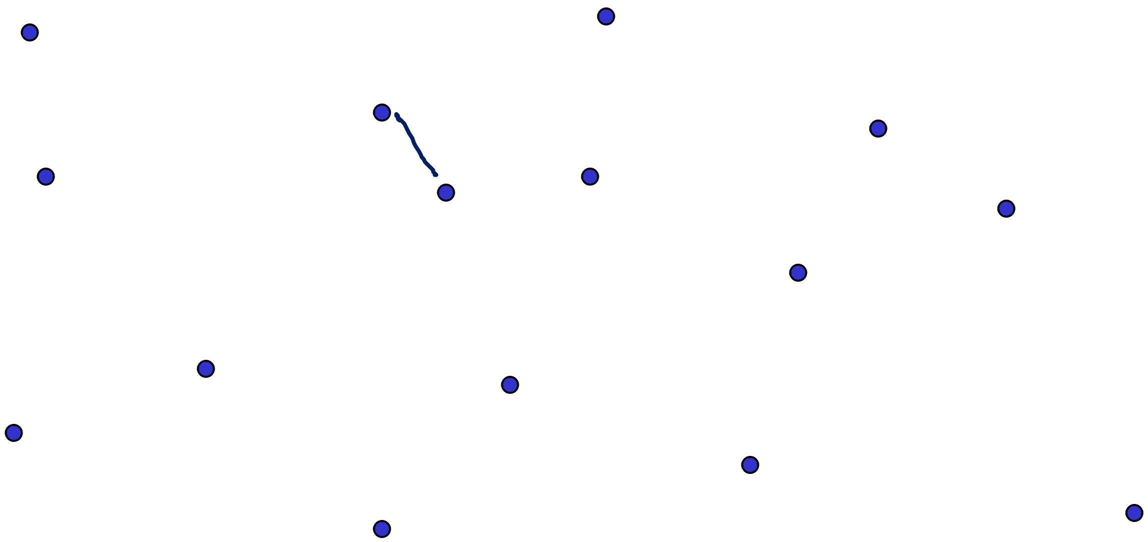
$$\overset{(\text{I.H.})}{\leq} 2\left(c\frac{n}{2}\left(\underbrace{1 + \log_2 \frac{n}{2}}_{\log_2 n}\right)\right) + cn$$

$$= \underline{c \cdot n \cdot \log_2 n + cn}$$

$\square$

Geometric divide-and-conquer

Closest Pair Problem: Given a set S of n points, find a pair of points with the smallest distance.

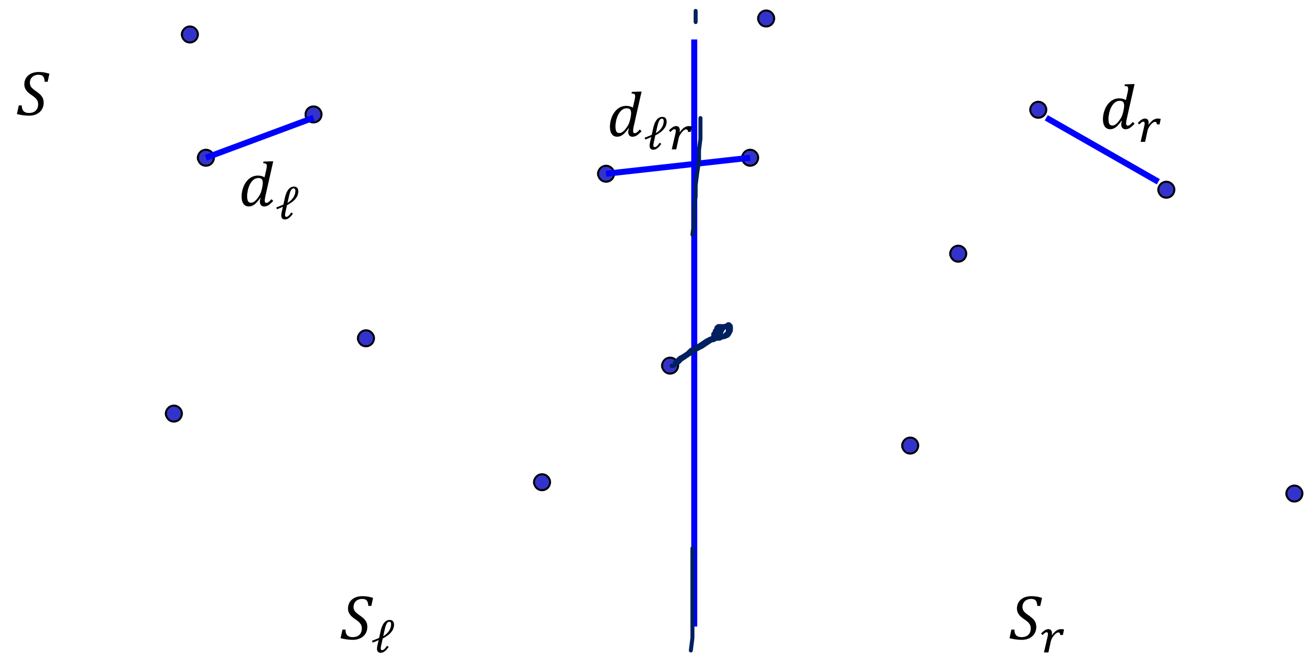


Naive solution:

go through all pairs: $\Theta(n^2)$

Divide-and-conquer solution

1. **Divide:** Divide S into two equal sized sets S_ℓ und S_r.
2. **Conquer:** $d_\ell = \mathrm{mindist}(S_\ell)$ $\quad$ $d_r = \mathrm{mindist}(S_r)$
3. **Combine:** $d_{\ell r} = \min\{d(p_\ell, p_r) \mid p_\ell \in S_\ell, p_r \in S_r\}$
   return $\min\{d_\ell, d_r, d_{\ell r}\}$

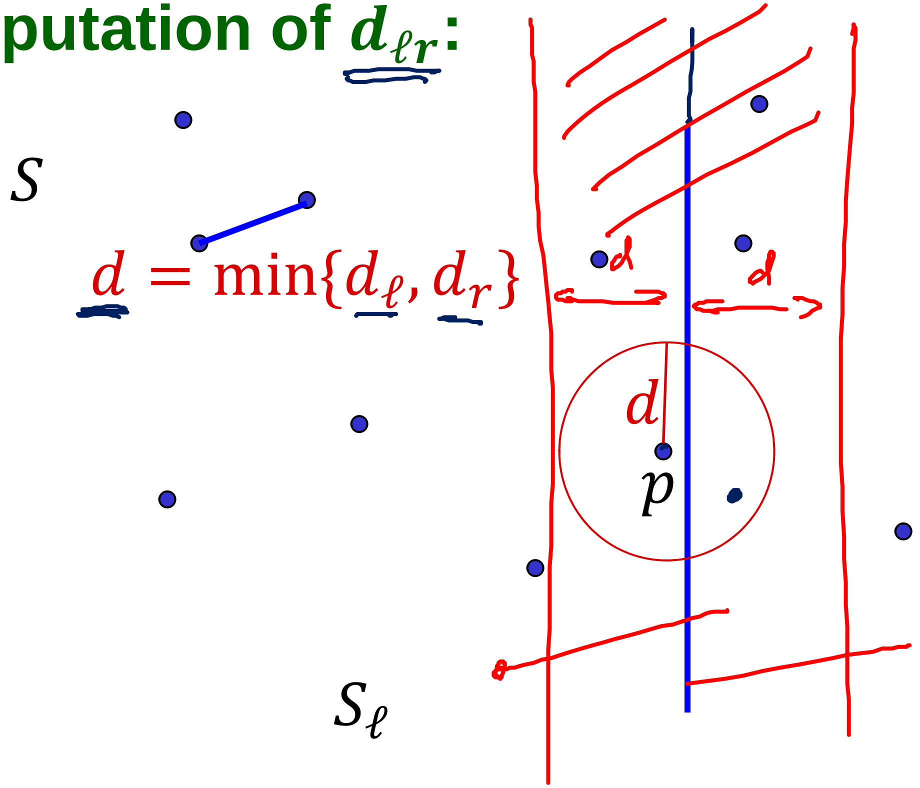


Divide-and-conquer solution

1. **Divide:** Divide S into two equal sized sets S_ℓ und S_r.
2. **Conquer:** $d_\ell = \text{mindist}(S_\ell)$ $d_r = \text{mindist}(Sr)$
3. **Combine:** $d_{\ell r} = \min\{d(p_\ell, p_r) \mid p_\ell \in S_\ell, p_r \in S_r\}$
   return $\min\{d_\ell, d_r, d_{\ell r}\}$

$d_{\ell r}$ only needed $d_{\ell r} < d$

Computation of $d_{\ell r}$:



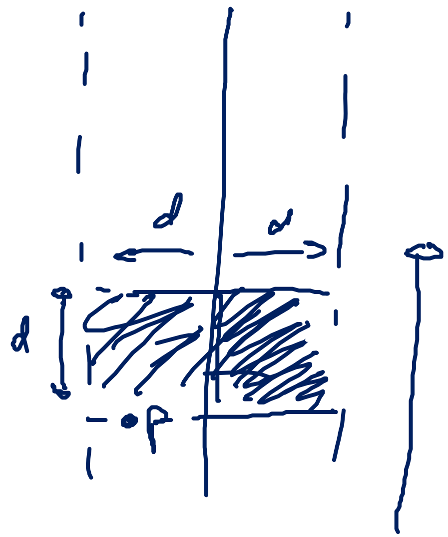
Merge step

1. Consider only points within distance $< d$ of the bisection line, in the order of increasing y-coordinates.

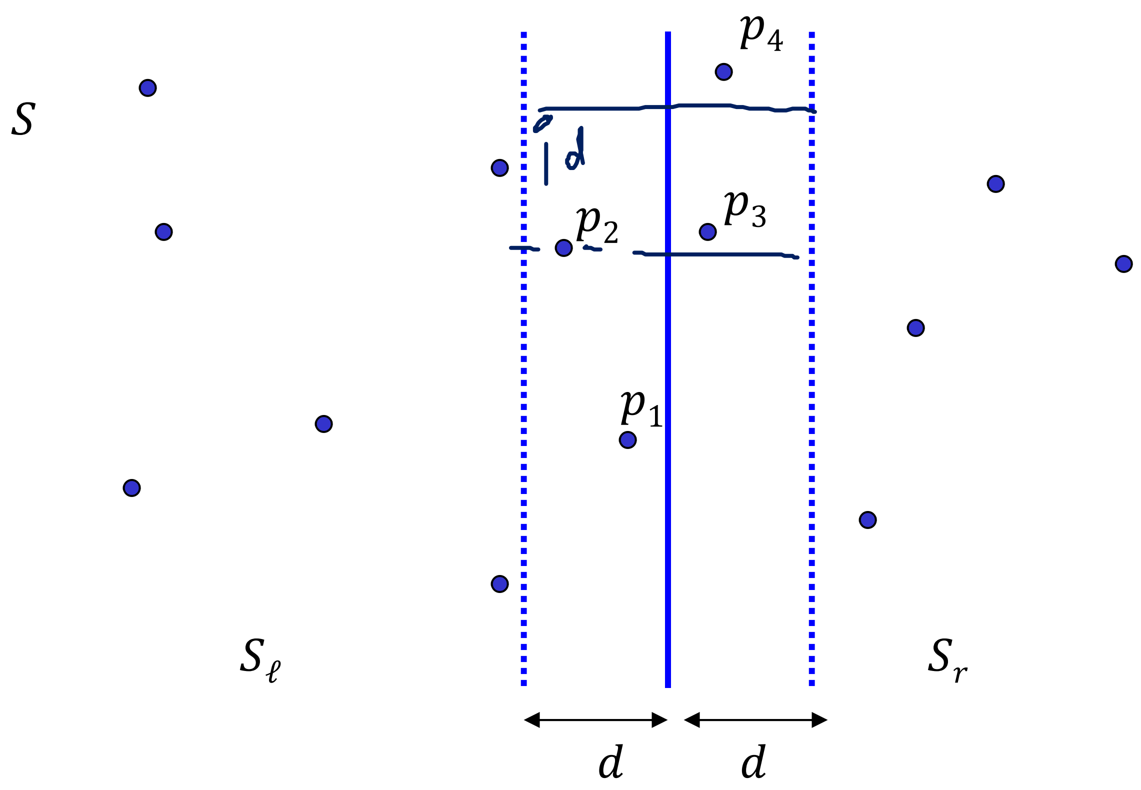
2. For each point p consider all points q with

$$\boldsymbol{y_p \leq y_q \leq y_p + d}$$

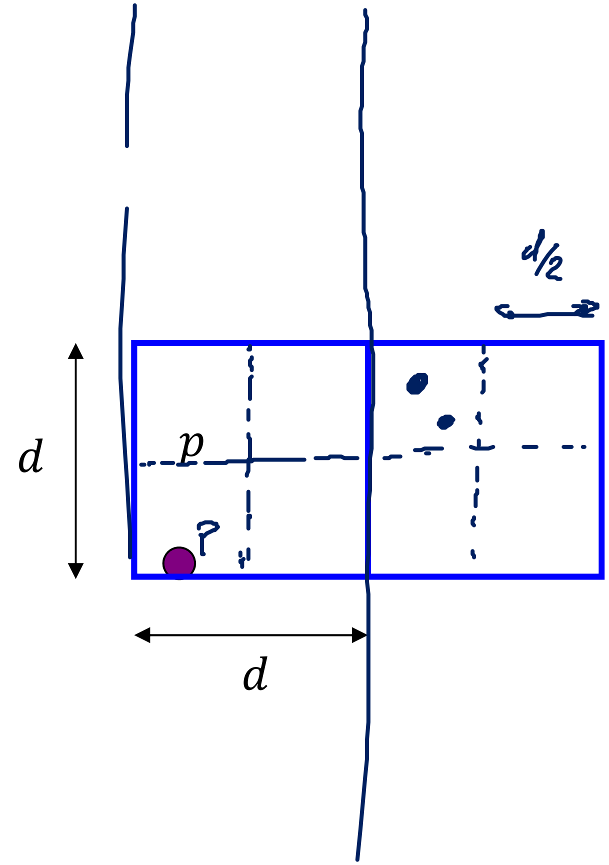
3. There are at most 7 such points.



Combine step



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



Implementation

- Initially sort the points in S in order of increasing x-coordinates

- While computing closest pair, also sort S according to y-coord.
  - Partition S into S_ℓ and S_r, solve and sort sub-problems recursively

    $2T(n/2)$

  - 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

    linear in n

Running Time

Recurrence relation:

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

Solution:

- Same as for computing number of number of inversions, merge sort (and many others…)

$$\underline{\underline{T(n) = O(n \cdot \log n)}}$$

Recurrence Relations: Master Theorem

Recurrence relation

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

Cases

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

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

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

  $$T(n) = \Theta\big(f(n)\big)$$

- $f(n) = \Theta\left(n^c \cdot \log^k n\right),\ c = \log_b a$ $\quad 2 \cdot T(n/2) + O(n)$

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