

Algorithmen und Datenstrukturen

Wintersemester 2019/20

17. Vorlesung

Nächstes Paar

Dr. Thomas van Dijk

~~*Prof. Dr. Alexander Wolff*~~

Lehrstuhl für Informatik I

Problem:

Gegeben: Menge P von n Punkten in der Ebene, jeder Punkt $p \in P$ als (x_p, y_p) .

Finde: Punktepaar $\{p, q\} \subseteq P$ mit kleinstem (euklidischen) Abstand.

Def.

Euklidischer Abstand von p und q ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

Lösung:

Laufzeit: $\Theta(n^2)$

- Gehe durch alle $\binom{n}{2}$ Punktepaare und berechne ihren Abstand.
- Gib ein Paar mit kleinstem Abstand zurück.

Mach's besser!

Entwurfsparadigma: – inkrementell?
– randomisiert?

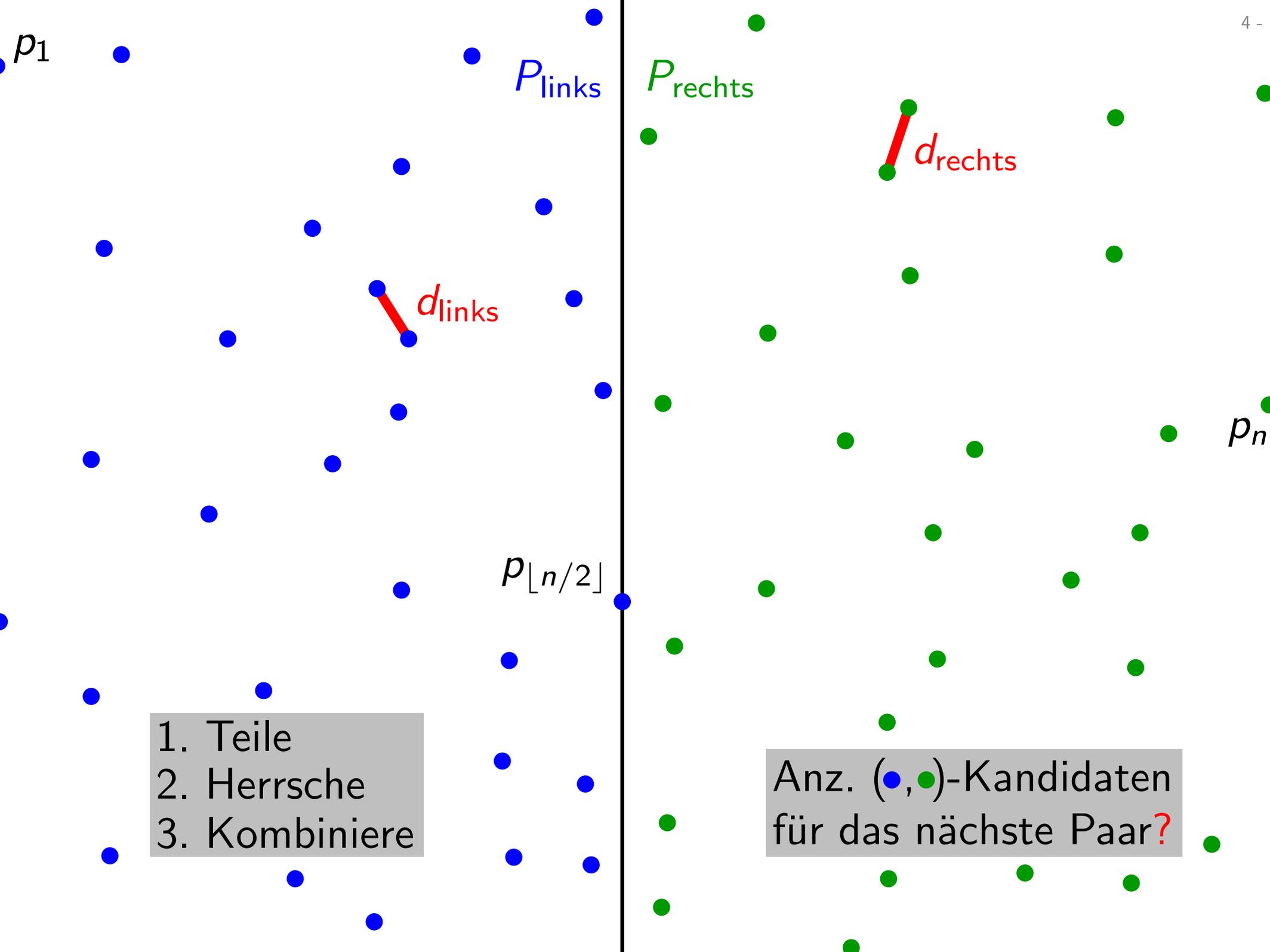
– Teile und Herrsche?!

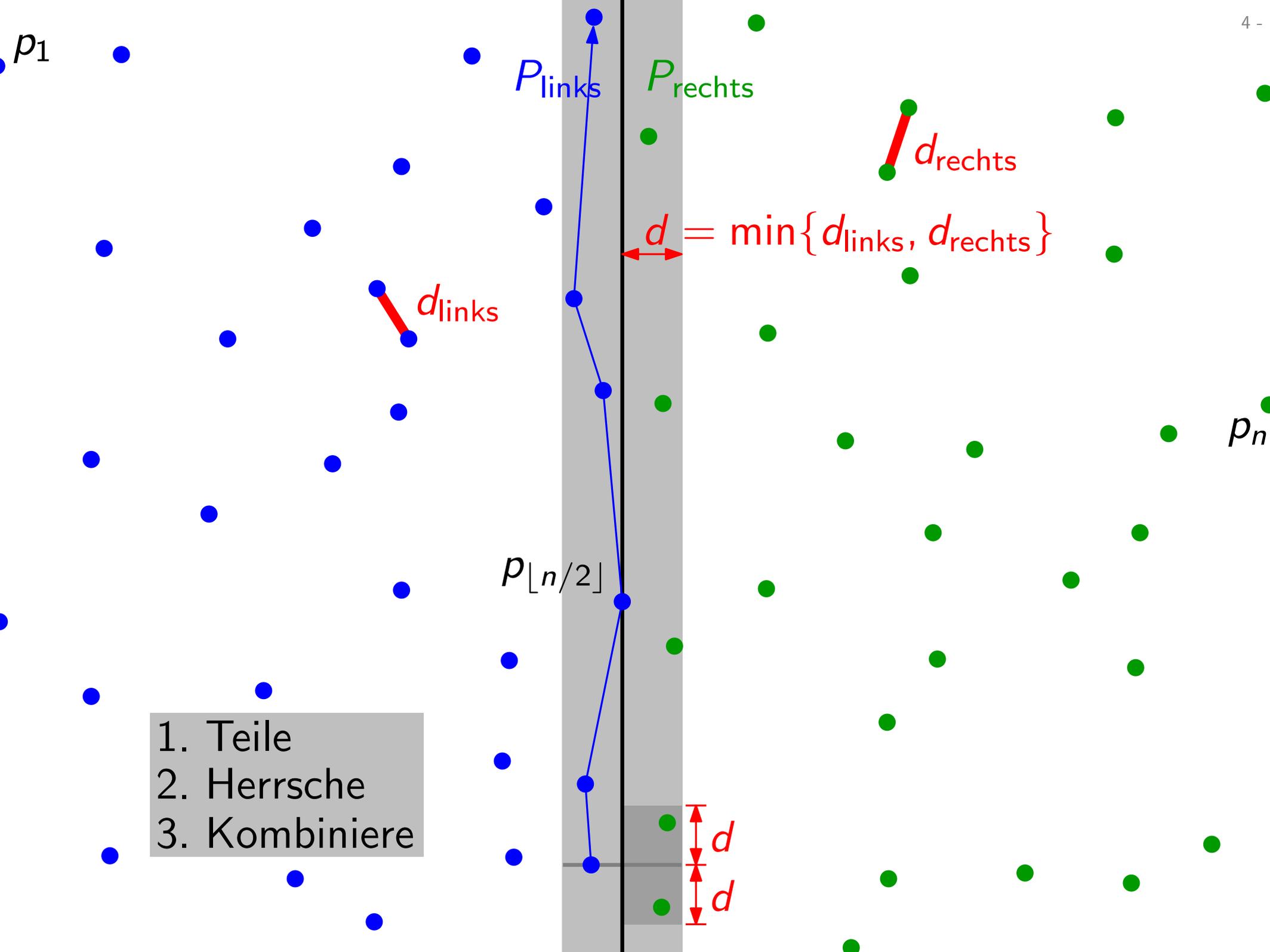
Spezialfall: 

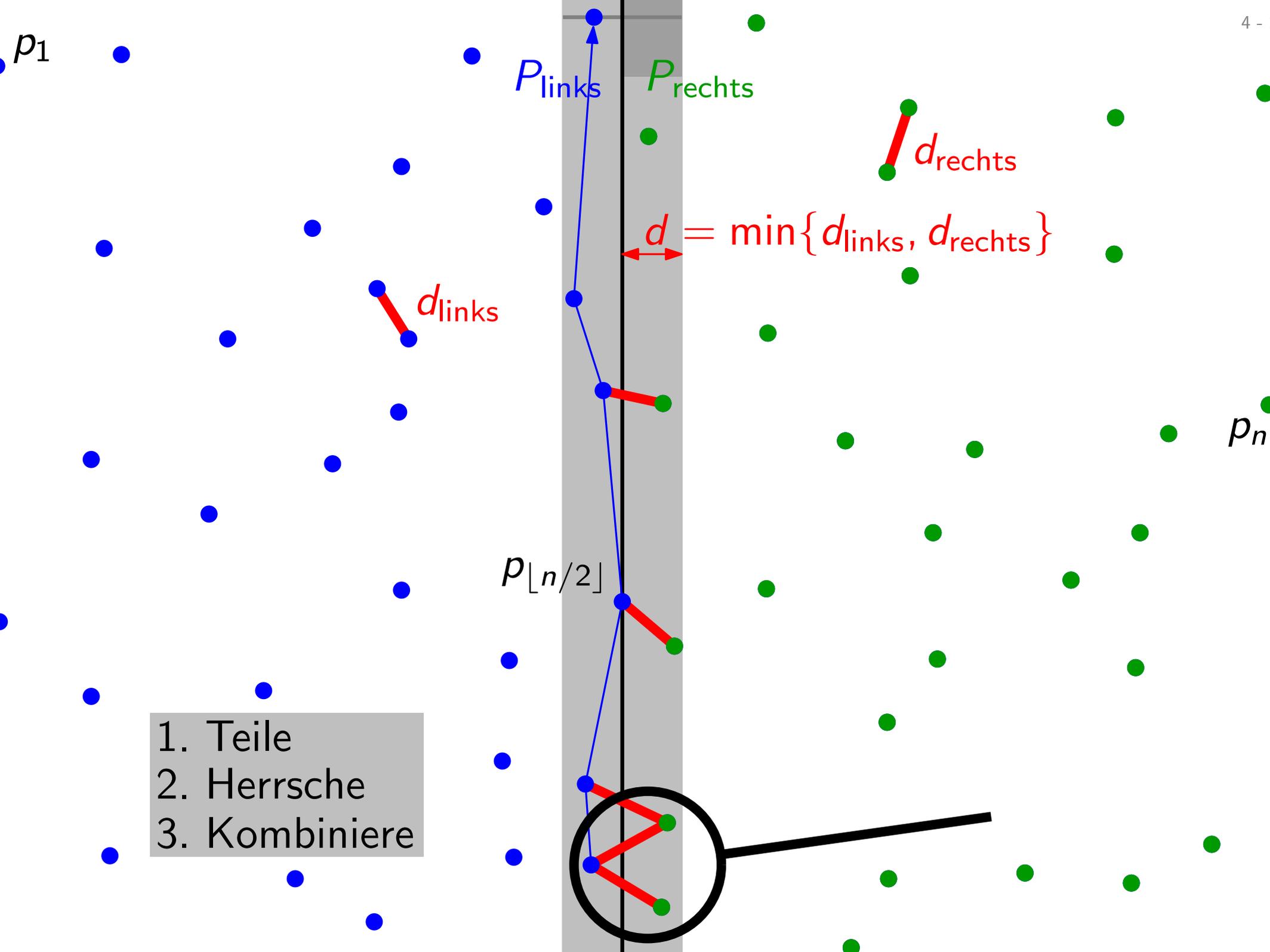
- Lösung:**
- Sortiere (nach x-Koordinate).
 - Berechne Abstände *aller aufeinanderfolgender* Punktepaare.
 - Bestimme das Minimum dieser Abstände.

Strukturelle Einsicht:

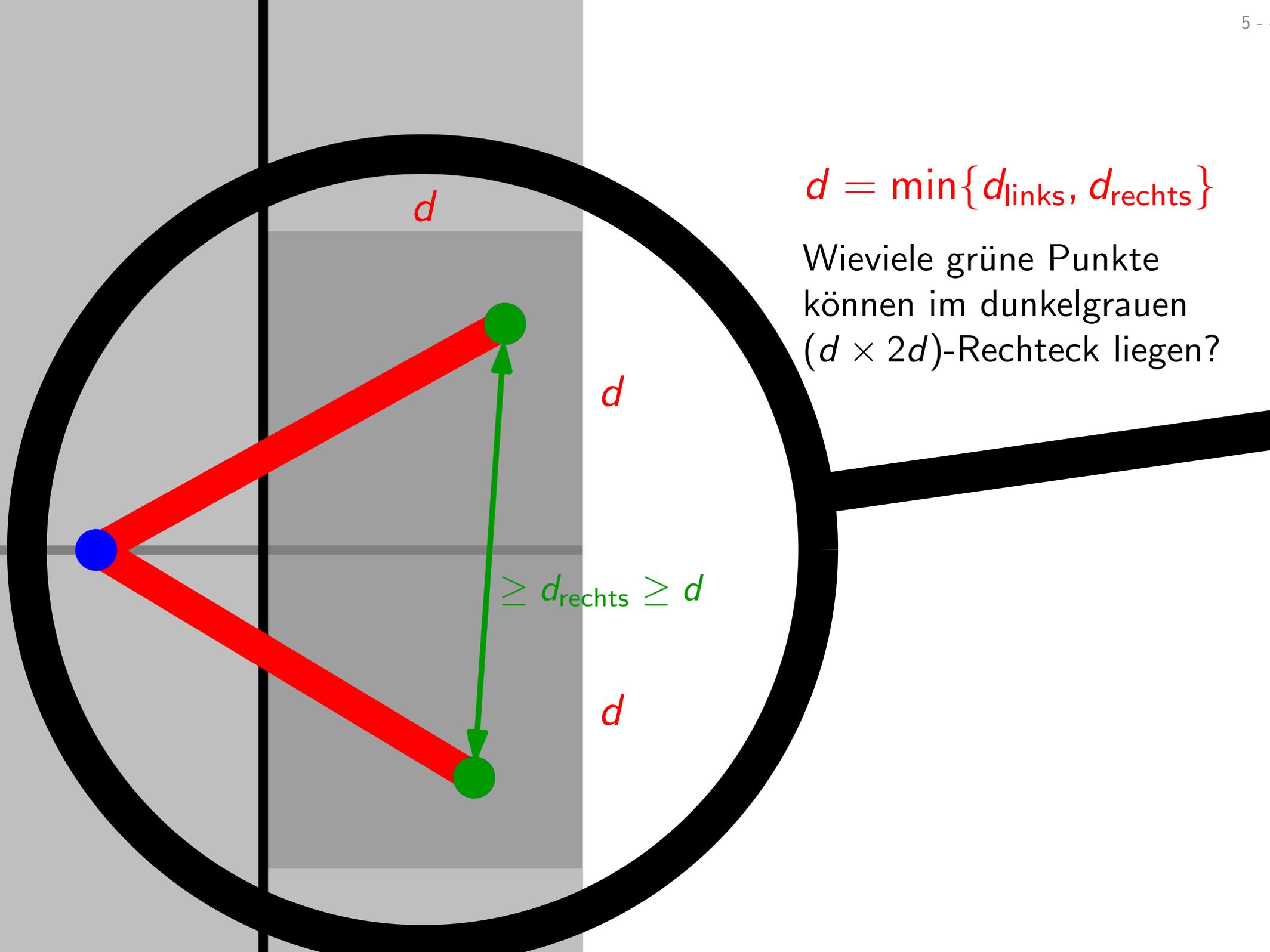
Kandidatenmenge der Größe $n - 1$,
die gesuchtes Objekt enthält.





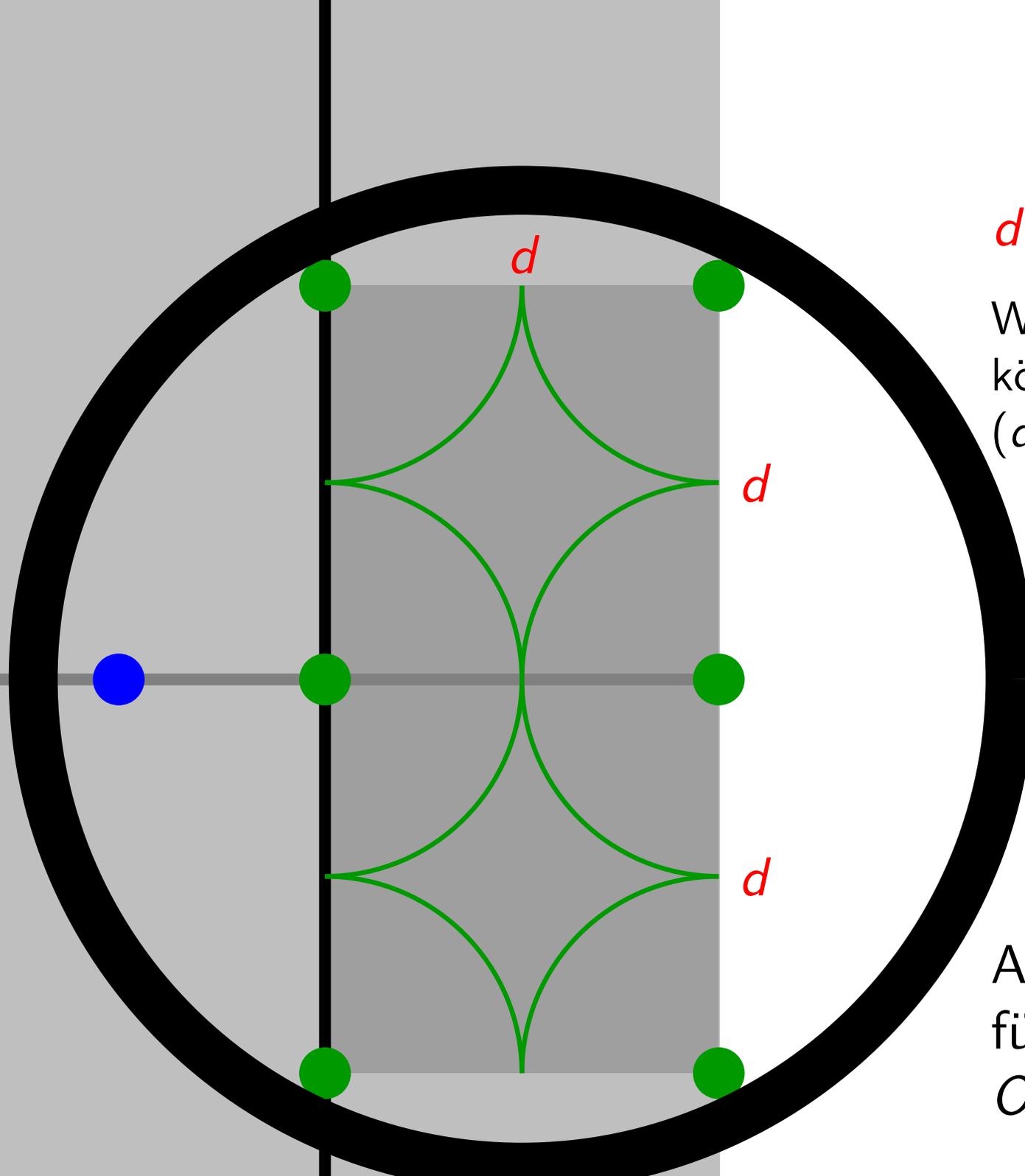


1. Teile
2. Herrsche
3. Kombiniere



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte
können im dunkelgrauen
($d \times 2d$)-Rechteck liegen?



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen $(d \times 2d)$ -Rechteck liegen?

Packungsargument:
maximal 6!



Anz. (\bullet, \bullet) -Kandidaten für das nächste Paar:
 $O(n)$. *Und finden?*

Algorithmus $T(n) =$ $\left\{ \begin{array}{l} \text{Laufzeit des rekursiven Teils,} \\ \text{d.h. ohne Vorverarbeitung (1.)} \end{array} \right.$

1. Sortiere P nach x-Koordinate $\rightarrow p_1, \dots, p_n$ mit $x_1 \leq \dots \leq x_n$

2. Teile: $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$, $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand d_{links} v. Paaren in P_{links}
 d_{rechts} P_{rechts}

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- sortiere P_{links} und P_{rechts} nach y-Koordinate
- gehe „gleichzeitig“ durch P_{links} und P_{rechts} :
 für jeden Punkt p in P_{links} gehe in P_{rechts} bis y-Koord. $y_p + d$;
 halte die letzten 6 Punkte im grauen Streifen aufrecht ($\rightarrow K_p$)
- bestimme Min. d_{mitte} über alle $d(p, q)$ mit $p \in P_{\text{links}}$ und $q \in K_p$
- gib Min. von d_{mitte} , d_{links} und d_{rechts} (und entspr. Paar) zurück

Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$

1. Sortiere P nach x-Koordinate $\rightarrow p_1, \dots, p_n$ mit $x_1 \leq \dots \leq x_n$

2. Teile: $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$, $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand d_{links} v. Paaren in P_{links}
 d_{rechts} P_{rechts}

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$ $O(1)$
- sortiere P_{links} und P_{rechts} nach y-Koordinate $O(n \log n)$
- gehe „gleichzeitig“ durch P_{links} und P_{rechts} :
 - für jeden Punkt p in P_{links} gehe in P_{rechts} bis y-Koord. $y_p + d$;
 - halte die letzten 6 Punkte im grauen Streifen aufrecht ($\rightarrow K_p$)
- bestimme Min. d_{mitte} über alle $d(p, q)$ mit $p \in P_{\text{links}}$ und $q \in K_p$
- gib Min. von d_{mitte} , d_{links} und d_{rechts} (und entspr. Paar) zurück

$O(n)$

Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte $n^{\log_b a} = n^{\log_2 2} = n^1$.

$$\text{Gilt } f \in \left\{ \begin{array}{l} O(n^{1-\varepsilon}) \quad \text{für ein } \varepsilon > 0 \\ \Theta(n^1) \\ \Omega(n^{1+\varepsilon}) \quad \text{für ein } \varepsilon > 0 \end{array} \right\} ?$$

Nein, $f: n \mapsto O(n \log n)$ passt in keinen der drei Fälle.



Die Rekursionsbaummethode liefert... $T(n) = O(n \log^2 n)$.

Noch besser?

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log^2 n)$$

1. Sortiere P nach x-Koordinate $\rightarrow p_1, \dots, p_n$ mit $x_1 \leq \dots \leq x_n$

2. Teile: P in $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ und $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

?! bestimme rekursiv kleinsten Abstand d_{links} v. Paaren in P_{links}
 d_{rechts} P_{rechts}

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$

- sortiere P_{links} und P_{rechts} nach y-Koordinate

$O(n)$ {

- gehe „gleichzeitig“ durch P_{links} und P_{rechts} :
für jeden Punkt p in P_{links} gehe in P_{rechts} bis y-Koord. $y_p + d$;
halte die letzten 6 Punkte im grauen Streifen aufrecht ($\rightarrow K_p$)
- bestimme Min. d_{mitte} über alle $d(p, q)$ mit $p \in P_{\text{links}}$ und $q \in K_p$
- gib Min. von d_{mitte} , d_{links} und d_{rechts} (und entspr. Paar) zurück

Noch besser!

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log n)$$

1. Sortiere P nach x-Koordinate $\rightarrow p_1, \dots, p_n$ mit $x_1 \leq \dots \leq x_n$
 und $P' = P$ nach y-Koordinate $\rightarrow p'_1, \dots, p'_n$ mit $y'_1 \leq \dots \leq y'_n$

2. Teile: P in $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ und $P_{\text{rechts}} = P \setminus P_{\text{links}}$
 P' in P'_{links} und P'_{rechts} (sortiert nach y-Koordinate)

3. Herrsche:

bestimme rekursiv kleinsten Abstand d_{links} v. Paaren in P_{links}
 d_{rechts} P_{rechts}

4. Kombiniere:

- $O(n)$ {
- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
 - gehe „gleichzeitig“ durch P'_{links} und P'_{rechts} :
 für jeden Punkt p in P'_{links} gehe in P'_{rechts} bis y-Koord. $y_p + d$;
 halte die letzten 6 Punkte im grauen Streifen aufrecht ($\rightarrow K_p$)
 - bestimme Min. d_{mitte} über alle $d(p, q)$ mit $p \in P'_{\text{links}}$ und $q \in K_p$
 - gib Min. von d_{mitte} , d_{links} und d_{rechts} (und entspr. Paar) zurück

Zusammenfassung

1. Vorverarbeitung ($2 \times$ Sortieren) $O(n \log n)$

2. Teilen $O(n)$

3. Herrschen $2T(n/2)$

4. Kombinieren $O(n)$

}

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

[MergeSort-Rek.!]

Gesamtlaufzeit

$O(n \log n)$



Speicherplatzbedarf?

$O(n)$, wenn P' *in situ* in P'_{links} und P'_{rechts} zerlegt wird.

Ist die Laufzeit $O(n \log n)$ optimal?

Def. *Element-Uniqueness-Problem (für natürliche Zahlen)*
 Gegeben eine Folge a_1, \dots, a_n von n Zahlen,
 kommt jede Zahl nur einmal vor, d.h. $a_i \neq a_j$ für $i \neq j$?

Satz. Das Element-Uniqueness-Problem kann nicht schneller
 als in $\Omega(n \log n)$ Zeit gelöst werden –
 wenn man als Rechenmodell das sogenannte
algebraische Entscheidungsbaummodell zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in $o(n \log n)$
 Zeit lösen – dann auch Element Uniqueness! ⚡

Wie? Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen a_1, \dots, a_n in eine Menge von (paarweise verschiedenen!)
 Punkten der Ebene transformieren, aber auch das geht! – Wie?



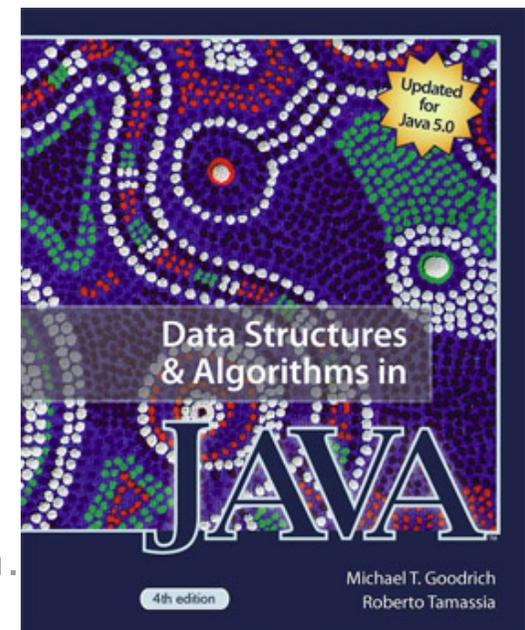
Das heißt. . .

- Satz.** Das Problem Nächstes Paar kann nicht schneller als in $\Omega(n \log n)$ Zeit gelöst werden, wenn man als Rechenmodell das algebraische Entscheidungsbaummodell zugrunde legt.
- Kor.** Unser $O(n \log n)$ -Zeit-Algorithmus für das Problem Nächstes Paar ist asymptotisch optimal, wenn man. . . .

УЧИТЬСЯ, УЧИТЬСЯ И УЧИТЬСЯ

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen) (●,●)-Kandidaten testet. *(Ist der schneller als der Brute-Force-Alg.?)*
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in $O(n \log^2 n)$ Zeit läuft!
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in $O(n \log n)$ Zeit läuft!

Goodrich & Tamassia:
Data Structures & Algorithms in Java.
Wiley, 4. Aufl., 2005 (5. Aufl., 2010)



Algorithmen & Datenstrukturen

Lernziele: In dieser Veranstaltung haben Sie schon gelernt...

- die Effizienz von Algorithmen zu **messen** und miteinander zu **vergleichen**,
- grundlegende Algorithmen und Datenstrukturen in Java zu **implementieren**,
- selbst Algorithmen und Datenstrukturen zu **entwerfen** sowie
- deren Korrektheit und Effizienz zu **beweisen**.

Inhalt:

- Grundlagen und Analysetechniken
- Sortierverfahren
- Java
- Datenstrukturen

To do

- Graphenalgorithmen (kürzeste Wege, min. Spannbäume)
- Systematisches Probieren (dynamisches Progr., Greedy-Alg.)