Divide-and-Conquer

Algorithmen und Datenstrukturen
VU 186.866, 5.5h, 8 ECTS, 2024S
Letzte Änderung: 11. April 2024
Vorlesungsfolien

Informatics

Algorithmen: Paradigmen

Greedy: Erstelle inkrementell eine Lösung, bei der nicht vorausschauend ein lokales Kriterium zur Wahl der jeweils nächsten hinzuzufügenden Lösungskomponente verwendet wird.

Divide-and-Conquer: Teile ein Problem in Teilprobleme auf. Löse jedes Teilproblem unabhängig und kombiniere die Lösung für die Teilprobleme zu einer Lösung für das ursprüngliche Problem.

Teile und Herrsche (*Divide-and-Conquer*)

Divide-and-Conquer: Allgemeines Prinzip, das häufig zu effizienten Problemlösungsstrategien führt.

Vorgehensweise:

- Teile Problem in mehrere Teile auf (meistens zwei).
- Löse jeden Teil rekursiv.
- Fasse Lösungen der Subprobleme zu einer Gesamtlösung zusammen.

> Divide et impera.
> Veni, vidi, vici.
> Julius Caesar

Mergesort

Sortieren: Wiederholung

Primitive Sortierverfahren:

- Bubblesort
- Insertionsort
- Selectionsort

Laufzeit: Die Laufzeit dieser Verfahren liegt im Worst- und Average-Case immer in $\Theta(n^2)$.

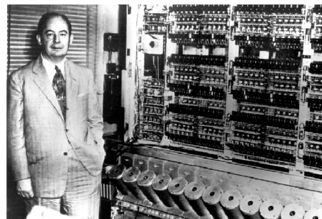
Frage: Kann man im Worst- und Average-Case schneller sortieren?

Antwort: Ja. Mergesort ist ein Beispiel dafür.

Mergesort (Sortieren durch Mischen)

Mergesort:

- Teile Array in zwei Hälften.
- Sortiere jede Hälfte rekursiv.
- Verschmelze zwei Hälften zu einem sortierten Ganzen.



John von Neumann (1945)

A L G O R I T H M S		
A L G O R I T H M S	teile	$O(1)$
A G L O R H I M S T	sortiere	$O(2T(n/2))$
A G H I L M O R S T	verschmelze	$O(n)$

Mergesort

Pseudocode:

- Mergesort für ein Array A.
- Sortiert den Bereich A[l] bis A[r].

```
Mergesort(A,l,r):
if l < r
    m ← ⌊(l + r)/2⌋
    Mergesort(A,l,m)
    Mergesort(A,m + 1,r)
    Merge(A,l,m,r)
```

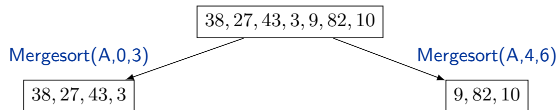
Aufruf: Mergesort(A, 0, $n - 1$) für ein Array A mit n Elementen.

Mergesort: Beispiel

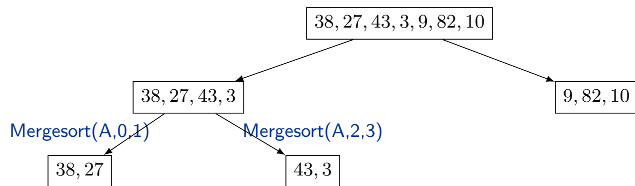
Mergesort(A,0,6)

$38, 27, 43, 3, 9, 82, 10$

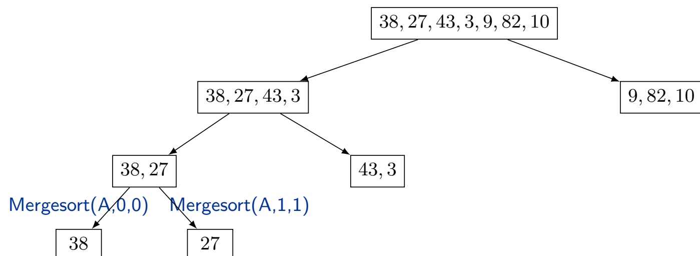
Mergesort: Beispiel



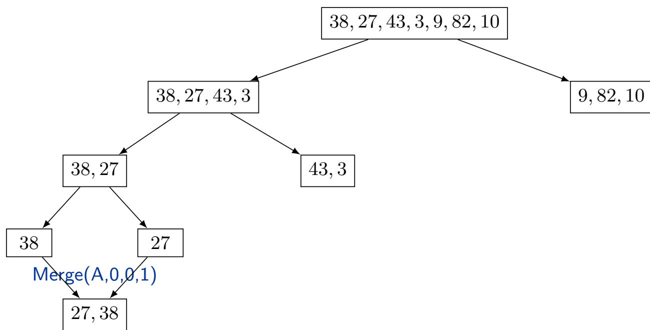
Mergesort: Beispiel



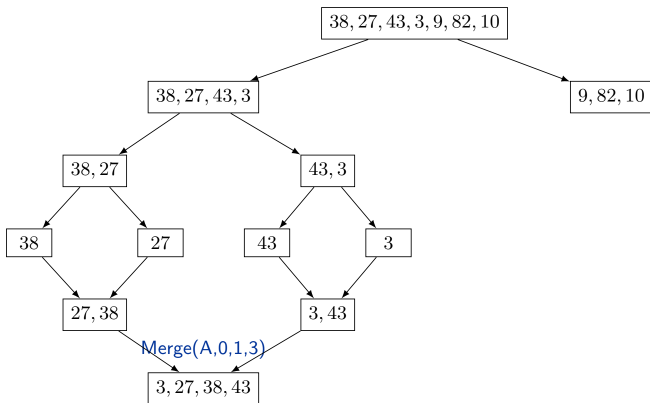
Mergesort: Beispiel



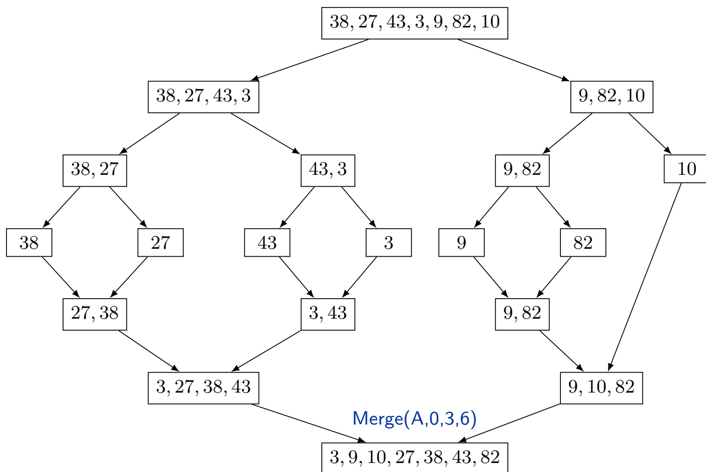
Mergesort: Beispiel



Mergesort: Beispiel



Mergesort: Beispiel

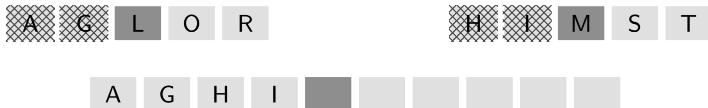


Merging (Verschmelzen)

Vorgehen: Verschmelze zwei sortierte Listen zu einer sortierten Gesamtliste.

Wie kann man effizient verschmelzen?

- Benutze temporäres Array.
- Durchlaufe beide Listen vom Anfang an.
- Führe die Elemente beider Listen im Reißverschlussverfahren zusammen, übernehme dabei jeweils das kleinste Element der beiden Listen.
- Hat lineare Laufzeit.



Verschmelzen

Pseudocode: Merge auf ein Array A. Verwendet Hilfsarray B.

```
Merge(A,l,m,r):
i ← l, j ← m+1, k ← l
while i ≤ m und j ≤ r
    if A[i] ≤ A[j]
        B[k] ← A[i], i ← i+1
    else
        B[k] ← A[j], j ← j+1
    k ← k+1
if i > m
    for h ← j bis r
        B[k] ← A[h], k ← k+1
else
    for h ← i bis m
        B[k] ← A[h], k ← k+1
for h ← l bis r
    A[h] ← B[h]
```

Eine nützliche Rekursionsgleichung

Definition: $C(n)$ = Anzahl der Schlüsselvergleiche (*comparisons*) in Mergesort bei einer Eingabegröße n.

Mergesort Rekursion:

$$C(n) \leq \begin{cases} 0 & \text{wenn } n = 1 \\ \underbrace{C\left(\lceil n/2 \rceil\right)}_{\text{linke Hälfte}} + \underbrace{C\left(\lfloor n/2 \rfloor\right)}_{\text{rechte Hälfte}} + \underbrace{n}_{\text{Verschmelzen}} & \text{sonst} \end{cases}$$

Lösung: $C(n) = O(n \log_2 n)$.

Eine nützliche Rekursionsgleichung

Beweis: Wir beschreiben mehrere Wege das $O(n \log_2 n)$ zu beweisen.

Annahmen:

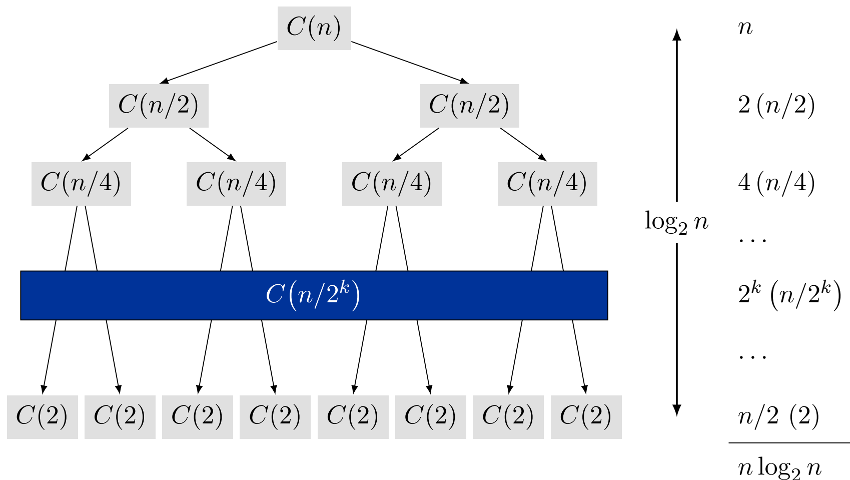
- Wir nehmen anfänglich an, dass n eine Zweierpotenz ist.
- Für ein allgemeines n' mit $\frac{n}{2} < n' < n$ (wobei n eine Zweierpotenz ist) gilt dann:

$$C(n') = O\left(n \log n\right)$$

da $O(\frac{n}{2} \log \frac{n}{2}) = O(n \log n)$ gilt.

Beweis durch Rekursionsbaum

$$C(n) \leq \begin{cases} 0 & \text{wenn } n = 1 \\ \underbrace{2C\left(n/2\right)}_{\text{Sortieren beider Hälften}} + \underbrace{n}_{\text{Verschmelzen}} & \text{sonst} \end{cases}$$



Beweis durch Auflösen der Rekursion

Behauptung: Wenn $C(n)$ die Rekursion erfüllt, dann $C(n) \leq n \log_2 n$.

$$C(n) \leq \begin{cases} 0 & \text{wenn } n = 1 \\ \underbrace{2C\left(n/2\right)}_{\text{Sortieren beider Hälften}} + \underbrace{n}_{\text{Verschmelzen}} & \text{sonst} \end{cases}$$

Beweis: Für $n > 1$:

$$\begin{aligned} \frac{C(n)}{n} &\leq \frac{2C(n/2)}{n} + 1 = \frac{C(n/2)}{n/2} + 1 \\ &\leq \frac{2C(n/4)}{n/2} + \frac{n/2}{n/2} + 1 = \frac{C(n/4)}{n/4} + 1 + 1 \\ &\leq \dots \\ &\leq \frac{C(n/n)}{n/n} + \underbrace{1 + \dots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Beweis durch Induktion

Behauptung: Wenn $C(n)$ die Rekursion erfüllt, dann $C(n) \leq n \log_2 n$.

$$C(n) \leq \begin{cases} 0 & \text{wenn } n = 1 \\ \underbrace{2C\left(n/2\right)}_{\text{Sortieren beider Hälften}} + \underbrace{n}_{\text{Verschmelzen}} & \text{sonst} \end{cases}$$

Beweis: (durch Induktion auf n)

- Induktionsanfang: $n = 1$.
- Induktionsbehauptung: $C(n) \leq n \log_2 n$.
- Ziel: Zeige, dass $C(2n) \leq 2n \log_2 2n$.

$$\begin{aligned} C(2n) &\leq 2C(n) + 2n \\ &\leq 2n \log_2 n + 2n \\ &= 2n(\log_2 n + 1) \\ &= 2n(\log_2(2n/2) + 1) \\ &= 2n(\log_2 2n - \log_2 2 + 1) \\ &= 2n(\log_2 2n - 1 + 1) \\ &= 2n \log_2 2n \end{aligned}$$

Analyse der Mergesort-Rekursion

Behauptung: Wenn $C(n)$ die folgende Rekursion erfüllt, dann $C(n) \leq n\lceil \log_2 n \rceil$.

$$C(n) \leq \begin{cases} 0 & \text{wenn } n = 1 \\ \underbrace{C(\lceil n/2 \rceil)}_{\text{linke Hälfte}} + \underbrace{C(\lfloor n/2 \rfloor)}_{\text{rechte Hälfte}} + \underbrace{n}_{\text{Verschmelzen}} & \text{sonst} \end{cases}$$

Beweis: (durch Induktion auf n)

- Induktionsanfang: $n = 1$.
- Definiere $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$.
- Induktionsschritt: Ist wahr für $1, 2, \ldots, n-1$.

$$\begin{aligned} C(n) &\leq C(n_1) + C(n_2) + n \\ &\leq n_1\lceil \log_2 n_1 \rceil + n_2\lceil \log_2 n_2 \rceil + n \\ &\leq n_1\lceil \log_2 n_1 \rceil + n_2\lceil \log_2 n_1 \rceil + n \\ &= n\lceil \log_2 n_1 \rceil + n \\ &\leq n\left(\lceil \log_2 n \rceil - 1\right) + n \\ &= n\lceil \log_2 n \rceil \end{aligned}$$

$$\begin{aligned} n_1 &= \lceil n/2 \rceil \\ &\leq \lceil 2^{\lceil \log_2 n \rceil}/2 \rceil \\ &= 2^{\lceil \log_2 n \rceil}/2 \\ &\Rightarrow \log_2 n_1 \leq \lceil \log_2 n \rceil - 1 \end{aligned}$$

Analyse der Mergesort-Rekursion

Asymptotische untere Schranke für $C(n)$: Das Verschmelzen für n Elemente benötigt zumindest $C_{\text{best}} = \lfloor \frac{n}{2} \rfloor$ Schlüsselvergleiche.

Daher gilt:

$$C_{\text{best}}(n) = C_{\text{worst}}(n) = C_{\text{avg}}(n) = \Theta(n \log n)$$

Allgemein: Die Laufzeit von Mergesort wird durch die Anzahl der notwendigen Vergleiche dominiert.

Daher gilt:

$$T_{\text{best}}(n) = T_{\text{worst}}(n) = T_{\text{avg}}(n) = \Theta(C(n)) = \Theta(n \log n)$$

Quicksort

Quicksort

Quicksort: Benutzt auch das Divide-and-Conquer-Prinzip, aber auf eine andere Art und Weise.

Teile: Wähle „Pivotelement“ x aus Folge A,
z.B. das an der letzten Stelle stehende Element.
Teile A ohne x so in zwei Teilfolgen A_1 und A_2, dass gilt:

- A_1 enthält nur Elemente $\leq x$.
- A_2 enthält nur Elemente $\geq x$.

Herrsche:

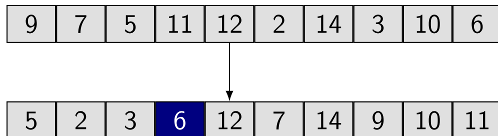
- Rekursiver Aufruf für A_1.
- Rekursiver Aufruf für A_2.

Kombiniere: Bilde A durch Hintereinanderfügen von A_1, x, A_2

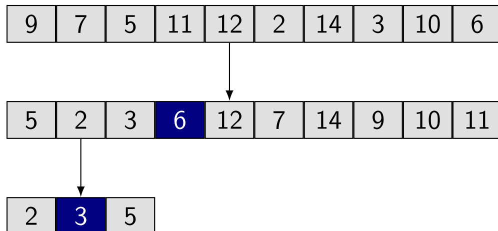
Quicksort: Beispiel

9	7	5	11	12	2	14	3	10	6

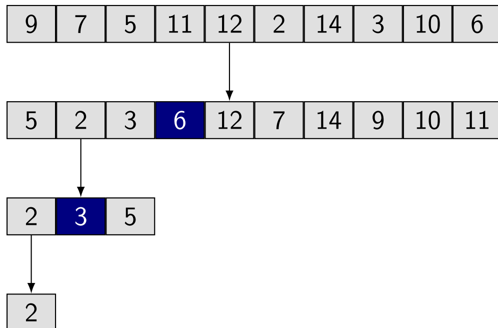
Quicksort: Beispiel



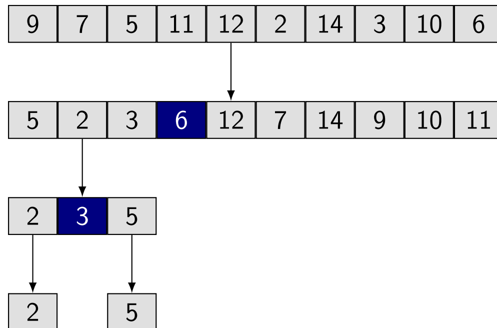
Quicksort: Beispiel



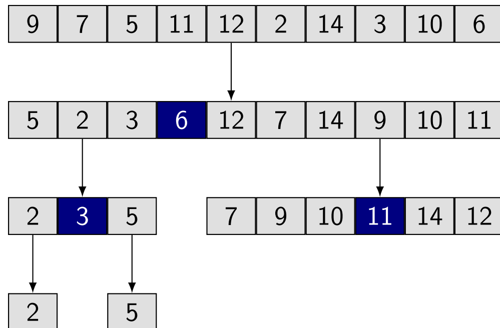
Quicksort: Beispiel



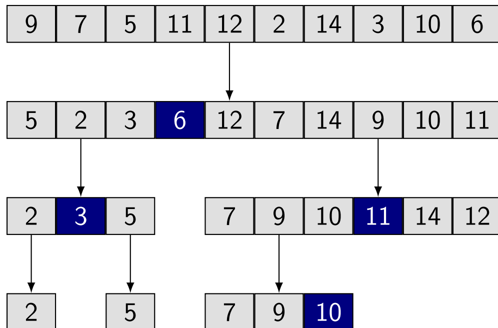
Quicksort: Beispiel



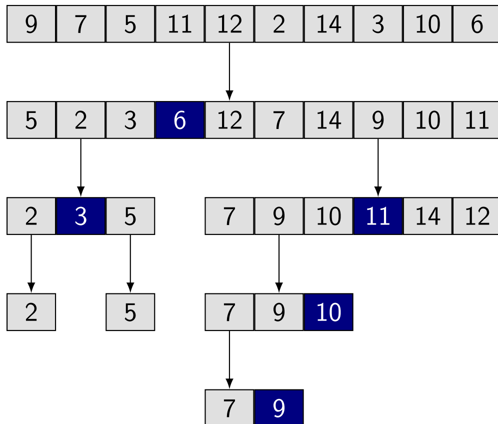
Quicksort: Beispiel



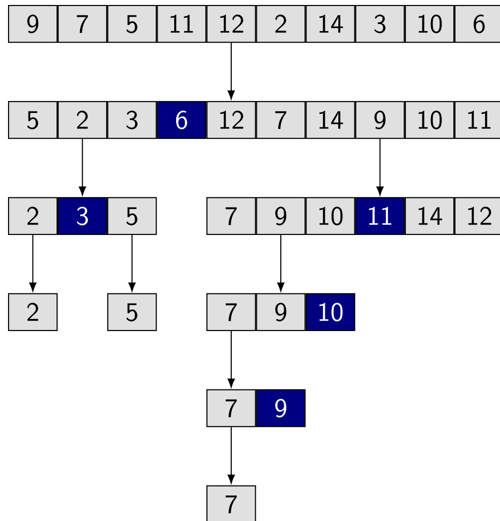
Quicksort: Beispiel



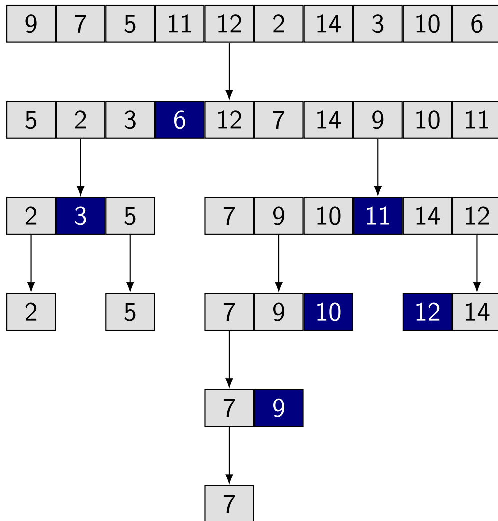
Quicksort: Beispiel



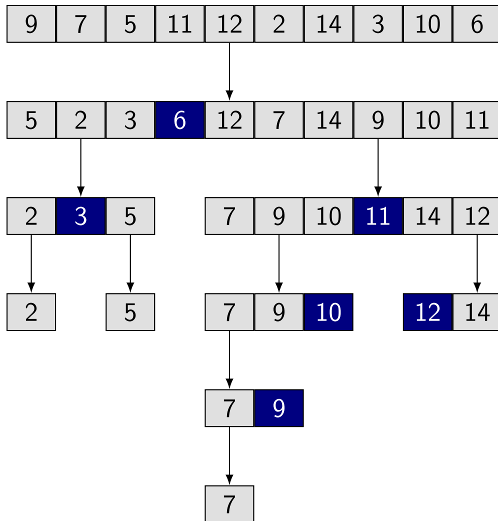
Quicksort: Beispiel



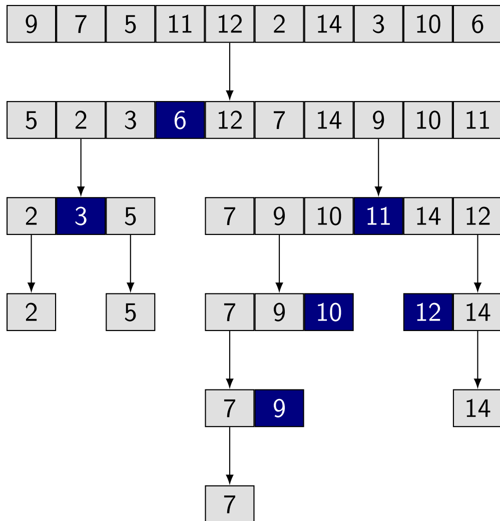
Quicksort: Beispiel



Quicksort: Beispiel



Quicksort: Beispiel



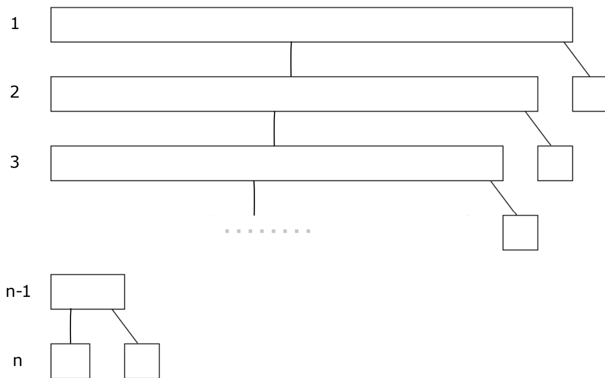
Quicksort: Analyse

Best-Case:

- Die beiden Teilfolgen haben immer (ungefähr) die gleiche Länge.
- Die Höhe des Aufrufbaumes ist $\Theta(\log n)$.
- Auf jeder Aufrufebene werden $\Theta(n)$ Vergleiche durchgeführt.
- Die Anzahl der Vergleiche und die Laufzeit liegen in $\Theta(n \log n)$.

Quicksort: Analyse

Worst-Case: Jede (Teil-)Folge wird immer beim letzten (oder immer beim ersten) Element geteilt.



Die Anzahl der Vergleiche ist $\Theta(n^2)$.

Quicksort: Analyse

Worst-Case: Mögliches Szenario:

- Aufsteigend sortierte Folge und es wird immer das hinterste Element als Pivotelement gewählt.
- Alle restlichen Elemente sind kleiner als das Pivotelement und daher wird die Rekursion nur für die linke Seite (alle restlichen Elemente außer dem Pivotelement) weitergeführt. Die rechte Seite ist leer (Terminierung der Rekursion).
- Die Größe der Teilfolge in der nächsten Rekursionsstufe verringert sich immer nur um 1!
- $\Theta(n)$ rekursive Aufrufe.
- Die Laufzeit liegt in $\Theta(n^2)$.
- Der zum Array zusätzlich benötigte Speicherplatz ist durch die $\Theta(n)$ rekursiven Aufrufe ebenfalls $\Theta(n)$.

Quicksort: Analyse

„Vermeiden“ des Worst-Case in der Praxis:

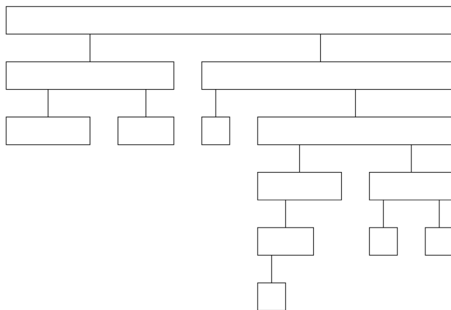
- Pivotelement immer zufällig wählen.
  - „Randomisierter Quicksort“.
  - Es ist nicht mehr die sortierte Folge das Worst-Case-Szenario.
- Betrachte jeweils das erste, letzte und mittlere Element (an der Postion $\lfloor \frac{n}{2} \rfloor$) und nimm den Median als Pivotelement.

Quicksort: Analyse

Average-Case:

- Komplizierter Beweis der zeigt, dass die Anzahl der Vergleiche und die Laufzeit dafür auch in $\Theta(n \log n)$ liegen.

Beispiel für Average-Case: Jede (Teil-)Liste wird nahe der Mitte geteilt.



Die Anzahl der Vergleiche ist $\Theta(n \log n)$.

Speicherplatzkomplexität

Speicherplatzkomplexität: Ist ein Maß für das Anwachsen des Speicherbedarfs eines Algorithmus in Abhängigkeit von der Eingabegröße.

Beispiele für Speicherplatzkomplexität:

- Quicksort: Worst-Case liegt in $\Theta(n)$, Best/Average-Case liegt in $\Theta(\log n)$
- Mergesort: Best/Average/Worst-Case liegt in $\Theta(n)$

Praxis: In der Praxis wird daher Quicksort sehr oft Mergesort vorgezogen.

Vergleich von Sortierverfahren

Tabelle: Laufzeit und Vergleiche.

Sortierverfahren	**Best-Case**	**Average-Case**	**Worst-Case**
Insertionsort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Selectionsort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Quicksort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$

Tabelle: Zusätzlicher Speicher.

Sortierverfahren	**Best-Case**	**Average-Case**	**Worst-Case**
Insertionsort	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Selectionsort	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Mergesort	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Quicksort	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n)$

Einsatz von Sortierverfahren

Quicksort:

- Wird sehr oft in allgemeinen Sortiersituationen bevorzugt.

Mergesort:

- Mergesort wird hauptsächlich für das Sortieren von Listen verwendet.
- Wird auch für das Sortieren von Dateien auf externen Speichermedien eingesetzt.
  - Dabei wird aber eine iterative Version von Mergesort (Bottom-up-Mergesort) verwendet, bei der nur $\log n$-mal eine Datei sequentiell durchgegangen wird.

Eine untere Schranke

Ausgangslage: Wir haben veschiedene Sortieralgorithmen kennengelernt. Die Worst-Case Laufzeit liegt im Bereich $O(n \log n)$ bis $O(n^2)$.

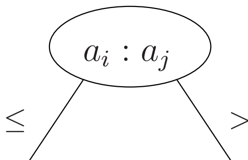
Frage: Geht es besser als in $O(n \log n)$ Zeit, z.B. $O(n)$?

Antwort: Wir zeigen für das allgemeine Sortierproblem unter der Annahme, dass n verschiedene Schlüssel sortiert werden müssen, dass $O(n \log n)$ optimal ist.

Entscheidungsbaum

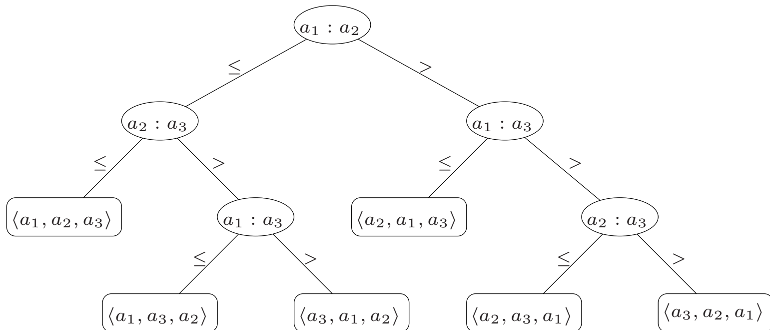
Entscheidungsbaum:

- Die Knoten entsprechen einem Vergleich von Elementen a_i und a_j.
- Die Blätter entsprechen den Permutationen.



Beispiel Entscheidungsbaum

Beispiel: Entscheidungsbaum des Insertionsort Algorithmus für $\langle a_1, a_2, a_3 \rangle$.



Beispiel Entscheidungsbaum

Anzahl der Schlüsselvergleiche: Die Anzahl der Schlüsselvergleiche im Worst-Case C_{worst} entspricht genau der Anzahl der Knoten auf dem längsten Pfad von der Wurzel bis zu einem Blatt minus 1.

Frage: Wie lautet die untere Schranke für die Höhe eines Entscheidungsbaums?

Satz: Jeder Entscheidungsbaum für die Sortierung von n paarweise verschiedenen Schlüsseln hat die Höhe $\Omega(n \log_2 n)$.

Entscheidungsbaum: Beweis

Beweis:

- Betrachte einen Entscheidungsbaum der Höhe h, der n unterschiedliche Schlüssel sortiert.
- Der Baum hat mindestens $n!$ Blätter.
- $n! \leq 2^h$, das impliziert $h \geq \log_2(n!)$.
- Hilfsrechnung:
  $n! \geq n \cdot (n-1) \cdot (n-2) \cdots \lceil \frac{n}{2} \rceil \geq (\lceil \frac{n}{2} \rceil)^{\lceil \frac{n}{2} \rceil} \geq \frac{n}{2}^{\frac{n}{2}}$
- $h \geq \log_2(n!) \geq \log_2 \left(\frac{n}{2}^{\frac{n}{2}} \right) \geq \frac{n}{2} \log_2 \frac{n}{2} = \frac{n}{2}(\log_2 n - 1) = \Omega(n \log_2 n)$

Untere Schranke für allgemeines Sortierproblem

Satz: Jedes allgemeine Sortierverfahren benötigt zum Sortieren von n paarweise verschiedenen Schlüsseln mindestens $\Omega(n \log n)$ Laufzeit im Worst-Case.

Folgerung: Mergesort ist ein asymptotisch zeitoptimaler Sortieralgorithmus. Es geht (asymptotisch) nicht besser!

Lineare Sortierverfahren

Beobachtung: Das Ergebnis für die untere Schranke basiert auf der Annahme, dass man keine Eigenschaften der zu sortierenden Elemente ausnutzt, sondern lediglich den Vergleichsoperator.

Praxis: Tatsächlich sind aber meist Wörter über ein bestimmtes Alphabet (d.h. einer bestimmten Definitionsmenge) gegeben, z.B.:

- Wörter über $\{a, b, \ldots, z, A, B, \ldots, Z\}$.
- Dezimalzahlen.
- Ganzzahlige Werte aus einem kleinen Bereich.

Lineare Sortierverfahren: Diese Information über die Art der Werte und ihren Wertebereich kann man zusätzlich ausnützen und dadurch im Idealfall Verfahren erhalten, die auch im Worst-Case in linearer Zeit sortieren.

Lineare Sortierverfahren: Beispiel Countsort

Eingabe: n Zahlen im Bereich 0 bis z im Array A, Hilfsarray Counts, $z < n$

```
for j ← 0 bis z
    Counts[j] ← 0
for i ← 0 bis n − 1
    Counts[A[i]] ← Counts[A[i]] + 1
i ← 0
for j ← 0 bis z
    for k ← 0 bis Counts[j] − 1
        A[i] ← j
        i ← i + 1
```

Komplexität: Erste Schleife in $\Theta(z)$, zweite Schleife in $\Theta(n)$, dritte (mit vierter) Schleife kann nur maximal n Zahlen bearbeiten und ist daher auch in $\Theta(n)$. Damit läuft der Algorithmus in $\Theta(z + n + n) = \Theta(n)$ (da $z = O(n)$).

Inversionen zählen

Inversionen zählen

Eine Musikwebsite versucht Übereinstimmungen im Musikgeschmack verschiedener Benutzer zu finden.

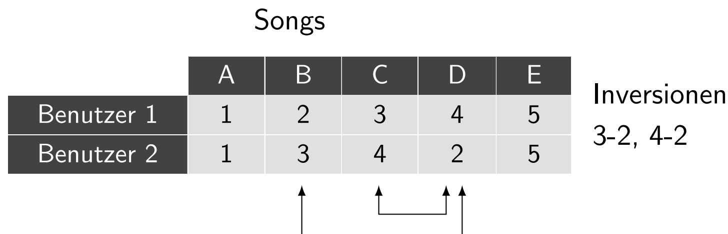
- Ein Benutzer bewertet n Songs indem er diese reiht.
- Die Musikseite überprüft in einer Datenbank, ob es weitere Benutzer mit einem ähnlichen Musikgeschmack gibt.

Ähnlichkeitsmaß: Anzahl der Inversionen zwischen zwei Rangfolgen.

- Rangfolge von Benutzer 1: $1, 2, \ldots, n$.
- Rangfolge von Benutzer 2: $a_1, a_2, \ldots, a_n$.
- Songs i und j sind invertiert, wenn $i < j$, aber $a_i > a_j$.

Inversionen zählen

Beispiel:



Brute-Force-Ansatz: Überprüfe alle $\Theta(n^2)$ Paare i und j.

Anwendungen

Anwendungen:

- Filtern von ähnlichen Präferenzen, wie z.B. ähnlichem Musikgeschmack (s.o.).
- Aggregieren von Rängen (z.B. für Metasuche im Web) durch finden einer Rangfolge, die eine minimale Anzahl an Inversionen im Vergleich zu den gegeben Rangfolgen hat.
- In der Statistik wird die Anzahl der Inversionen zwischen zwei Rangfolgen für den Tau-Test, einen nichtparametrischen Hypothesentest, benutzt.

Inversionen zählen: Divide-and-Conquer

Divide-and-Conquer:

1	5	4	8	10	2	6	9	12	11	3	7

Inversionen zählen: Divide-and-Conquer

Divide-and-Conquer:

- Divide: Teile Liste in zwei Teile.

Divide: $O(1)$

Inversionen zählen: Divide-and-Conquer

Divide-and-Conquer:

- Divide: Teile Liste in zwei Teile.
- Conquer: Zähle rekursiv die Anzahl der Inversionen in jeder Hälfte.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: $2T(n/2)$

5 blau-blau Inversionen

5-4, 5-2, 4-2, 8-2, 10-2

8 rot-rot Inversionen

6-3, 9-3, 9-7, 12-3,
12-7, 12-11, 11-3, 11-7

Inversionen zählen: Divide-and-Conquer

Divide-and-Conquer:

- Divide: Teile Liste in zwei Teile.
- Conquer: Zähle rekursiv die Anzahl der Inversionen in jeder Hälfte.
- Combine: Zähle Inversionen, wobei a_i und a_j sich in unterschiedlichen Hälften befinden, und retourniere die Summe von drei Größen.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$

1	5	4	8	10	2

5 blau-blau Inversionen

6	9	12	11	3	7

8 rot-rot Inversionen

Conquer: $2T(n/2)$

9 blau-rot Inversionen

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Gesamt $= 5 + 8 + 9 = 22$.

Inversionen zählen: Kombinieren

Kombinieren: Zähle blau-rot Inversionen.

- Es wird angenommen, dass jede Hälfte sortiert ist.
- Zähle Inversionen, wobei a_i und a_j sich in unterschiedlichen Hälften befinden.
- Verschmelze zwei sortierte Hälften in eine sortierte Folge.
  - *Um die Invariante der Sortierung aufrechtzuerhalten*

9 blau-rot Inversionen: 4+2+2+1+0+0

Zählen: $O(n)$

Verschmelzen: $O(n)$

Inversionen zählen: Implementierung

Precondition: [Merge-and-Count] A und B sind sortiert. Postcondition: [Sort-and-Count] L ist sortiert.

```
Sort-and-Count(L):
if Liste L hat genau ein Element
    return 0 und die Liste L

Unterteile die Liste in zwei Hälften A und B
(r_A, A)  ← Sort-and-Count(A)
(r_B, B)  ← Sort-and-Count(B)
(r, L)    ← Merge-and-Count(A, B)

return r_A + r_B + r und die sortierte Liste L
```

Inversionen zählen: Implementierung

```
Merge-and-Count(A,B):
i ← 1,  j ← 1
count ← 0
while beide Listen sind nicht leer
    if a_i ≤ b_j
        Füge a_i zur Ergebnisliste hinzu und erhöhe i
    else
        Füge b_j zur Ergebnisliste hinzu und erhöhe j
        count ← count + die Anzahl der restlichen Elemente in A
Füge den Rest der nicht leeren Liste zur Ergebnisliste hinzu
return count und sortierte Folge (Verschmelzung beider Hälften)
```

Inversionen zählen: Analyse

Verschmelzen: Merge-And-Count läuft in $O(n)$
(wie Merge-Schritt bei Mergesort).

Laufzeit: Die Laufzeit von Sort-and-Count lässt sich dann beschreiben mit:

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

Hinweis: Sort-and-Count ist nichts anderes als ein Mergesort, der noch zusätzlich Inversionen zählt.

Dichtestes Punktpaar (*Closest Pair of Points*)

Dichtestes Punktpaar

Dichtestes Paar: Gegeben seien n Punkte in der Ebene. Finde ein Paar mit der kleinsten euklidischen Distanz zwischen den beiden Punkten.

Fundamentale geometrische Formen:

- Grafik, maschinelles Sehen, geographische Informationssysteme, molekulare Modellierung, Flugsicherung.
- Spezialfall von nächster Nachbar, Euklidischer MST, Voronoi.

  ☐ *Schnelle Algorithmen für das dichteste Punktpaar waren die Inspiration für schnelle Algorithmen für dieses Problem.*

Brute-Force-Ansatz: Überprüfe alle Paare von Punkten p und q mit $\Theta(n^2)$ Vergleichen.

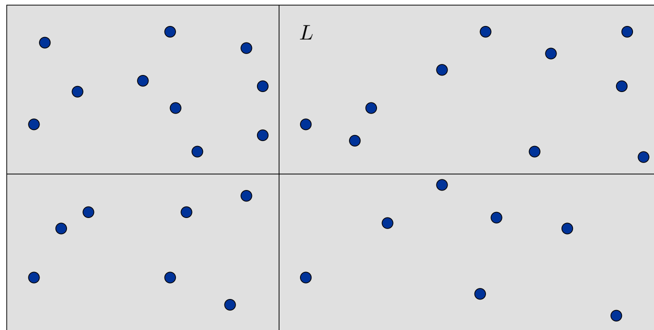
1-D-Version: $O(n \log n)$, wenn sich alle Punkte auf einer Linie befinden.

Annahme: Zwei Punkte haben nicht dieselbe x-Koordinate.

☐ *um Darstellung sauberer zu machen.*

Dichtestes Punktpaar: Ein erster Versuch

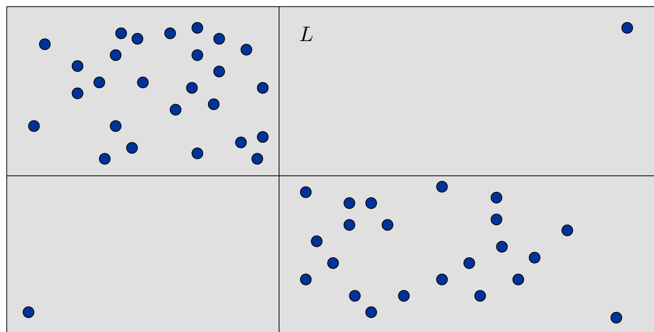
Teile: Teile die Region in 4 Quadranten.



Dichtestes Punktpaar: Ein erster Versuch

Teile: Teile die Region in 4 Quadranten.

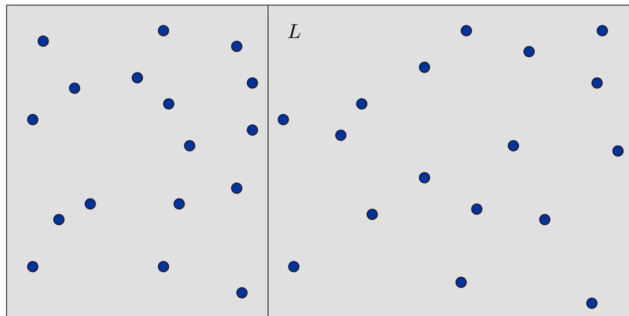
Problem: Es ist i.A. nicht möglich $n/4$ Punkte in jedem Quadranten sicherzustellen.



Dichtestes Punktpaar

Algorithmus:

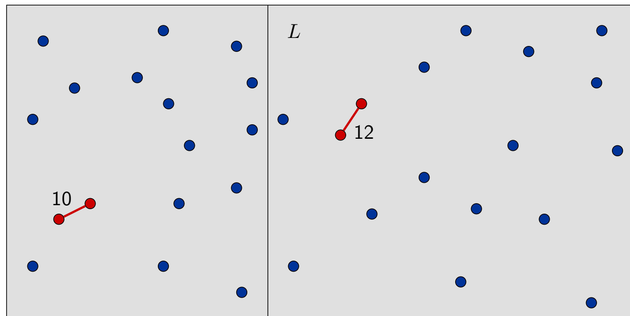
- Teile: Zeichne eine vertikale Linie L, sodass sich ungefähr $\frac{1}{2}n$ Punkte auf jeder Seite befinden.



Dichtestes Punktpaar

Algorithmus:

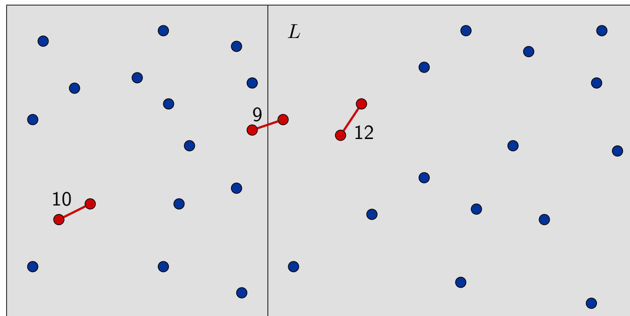
- Teile: Zeichne eine vertikale Linie L, sodass sich ungefähr $\frac{1}{2}n$ Punkte auf jeder Seite befinden.
- Herrsche: Finde ein dichtestes Punktpaar auf jeder Seite rekursiv.



Dichtestes Punktpaar

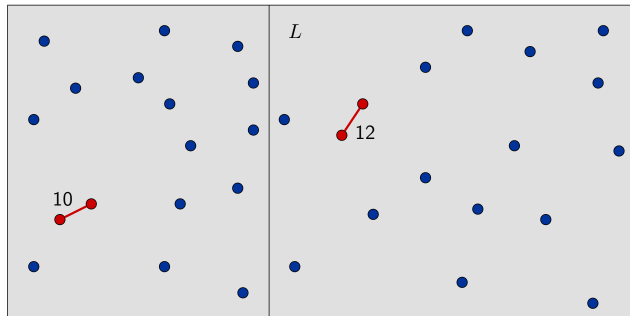
Algorithmus:

- Teile: Zeichne eine vertikale Linie L, sodass sich ungefähr $\frac{1}{2}n$ Punkte auf jeder Seite befinden.
- Herrsche: Finde ein dichtestes Punktpaar auf jeder Seite rekursiv.
- Kombiniere: Finde ein dichtestes Punktpaar mit einem Punkt auf jeder Seite (scheint in $\Theta(n^2)$ zu liegen).
- Retourniere die Beste von den drei Lösungen.



Dichtestes Punktpaar

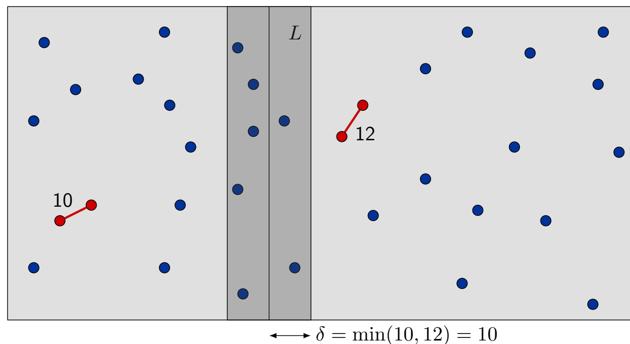
Aufgabe: Finde das dichteste Punktpaar mit einem Punkt auf jeder Seite unter der Annahme, dass die Distanz $< \delta$ ($=$ Minimum der minimalen Punktabstände in der linken und rechten Hälfte) ist.



Dichtestes Punktpaar

Aufgabe: Finde das dichteste Punktpaar mit einem Punkt auf jeder Seite unter der Annahme, dass die Distanz $< \delta$ (= Minimum der minimalen Punktabstände in der linken und rechten Hälfte) ist.

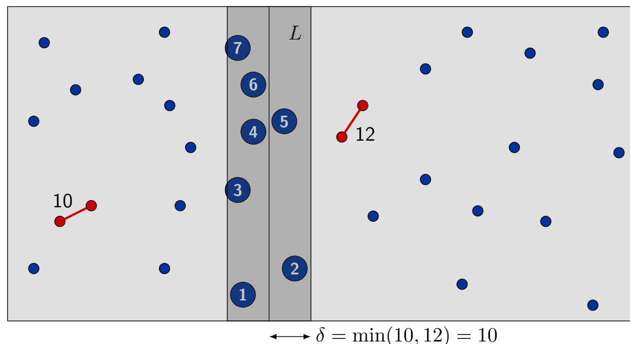
- Beobachtung: Man muss nur die Punkte mit einem Abstand kleiner δ von L berücksichtigen.



Dichtestes Punktpaar

Aufgabe: Finde das dichteste Punktpaar mit einem Punkt auf jeder Seite unter der Annahme, dass die Distanz $< \delta$ ($=$ Minimum der minimalen Punktabstände in der linken und rechten Hälfte) ist.

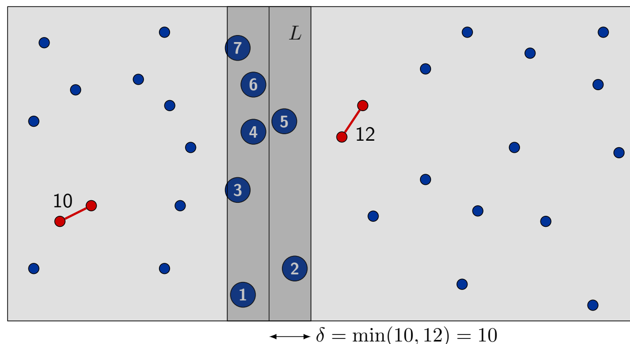
- Beobachtung: Man muss nur die Punkte mit einem Abstand kleiner δ von L berücksichtigen.
- Sortiere Punkte im 2δ-Streifen nach ihren y-Koordinaten.



Dichtestes Punktpaar

Aufgabe: Finde das dichteste Punktpaar mit einem Punkt auf jeder Seite unter der Annahme, dass die Distanz $< \delta$ ($=$ Minimum der minimalen Punktabstände in der linken und rechten Hälfte) ist.

- Beobachtung: Man muss nur die Punkte mit einem Abstand kleiner δ von L berücksichtigen.
- Sortiere Punkte im 2δ-Streifen nach ihren y-Koordinaten.
- Man muss nur die Distanzen von jedem dieser Punkte zu max. 11 in der sortierten Liste nachfolgenden Punkten überprüfen!



Dichtestes Punktpaar

Definition: Sei s_i der Punkt im 2δ-Streifen mit der i-ten kleinsten y-Koordinate.

Behauptung: Wenn $|i-j| \geq 12$, dann ist die Distanz zwischen s_i und s_j zumindest δ.

Beweis:

- Keine zwei Punkte liegen im gleichen $\frac{1}{2}\delta$-mal-$\frac{1}{2}\delta$ Bereich.
- Zwei Punkte, die zumindest 2 ganze Zeilen von einander entfernt sind, haben eine Distanz von $\geq 2\left(\frac{1}{2}\delta\right)$. □

Fakt (ohne Beweis): Noch immer wahr wenn wir 12 durch 7 ersetzen.

Dichtestes Punktpaar

```
Dichtestes Paar(p_1, ..., p_n):
    if n = 1 return ∞

    Berechne Trennlinie L, sodass Punkte auf zwei Hälften
    aufgeteilt werden.                                              O(n log n)

    δ_1 = Dichtestes Paar(linke Hälfte)
    δ_2 = Dichtestes Paar(rechte Hälfte)                            2T(n/2)
    δ   = min(δ_1, δ_2)

    Lösche alle Punkte die weiter als δ von der
    Trennlinie L entfernt sind                                      O(n)

    Sortiere die restlichen Punkte nach y-Koordinate.               O(n log n)

    Scanne die Punkte in y-Reihenfolge und vergleiche
    die Distanz zwischen jedem Punkt und den nächsten
    11 Nachbarn. Wenn eine dieser Distanzen kleiner als δ           O(n)
    ist, ändere δ

    return δ.
```

Laufzeiten: Trennlinie $O(n \log n)$; Rekursion $2T(n/2)$; Löschen $O(n)$; Sortieren $O(n \log n)$; Scannen $O(n)$.

Dichtestes Punktpaar: Analyse

Laufzeit:

$$T(n) \leq 2T(n/2) + O(n \log n) \quad \Rightarrow \quad T(n) = O(n \log^2 n)$$

Frage: Können wir $O(n \log n)$ erreichen?

Antwort: Ja. Die Punkte im Streifen müssen nicht immer sortiert werden.

- Jede Rekursion liefert zwei Listen zurück: Alle Punkte sortiert nach y-Koordinate, und alle Punkte sortiert nach x-Koordinate.
- Sortiere durch Verschmelzen zweier vorsortierter Listen.

$$T(n) \leq 2T(n/2) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)$$