IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(25 – Sortieren vorsortierter Daten)

Untere Schranke für allgemeine Sortierverfahren

Satz

Zum Sortieren einer Folge von n Schlüsseln mit einem allgemeinen Sortierverfahren sind im Worst-Case ebenso wie im Mittel wenigstens $\Omega(n \log n)$ Vergleichsoperationen zwischen zwei Schlüsseln erforderlich.

Modellierung von allgemeinen Sortierverfahren:

Mit Hilfe von Entscheidungsbäumen.

Entscheidungsbäume

Gegeben ein allgemeines Sortierverfahren A:

Entscheidungsbaum $T_{A,n}$ zur Modellierung des Ablaufs von A auf Folgen der Länge n enthält:

- Für jede der $n!$ Permutationen ein Blatt
- Innere Knoten repräsentieren eine Vergleichsoperation und haben zwei Söhne
- Weg W von der Wurzel zu einem Blatt v:

  Die Vergleiche an den Knoten von W identifizieren die Permutation π_v von v,

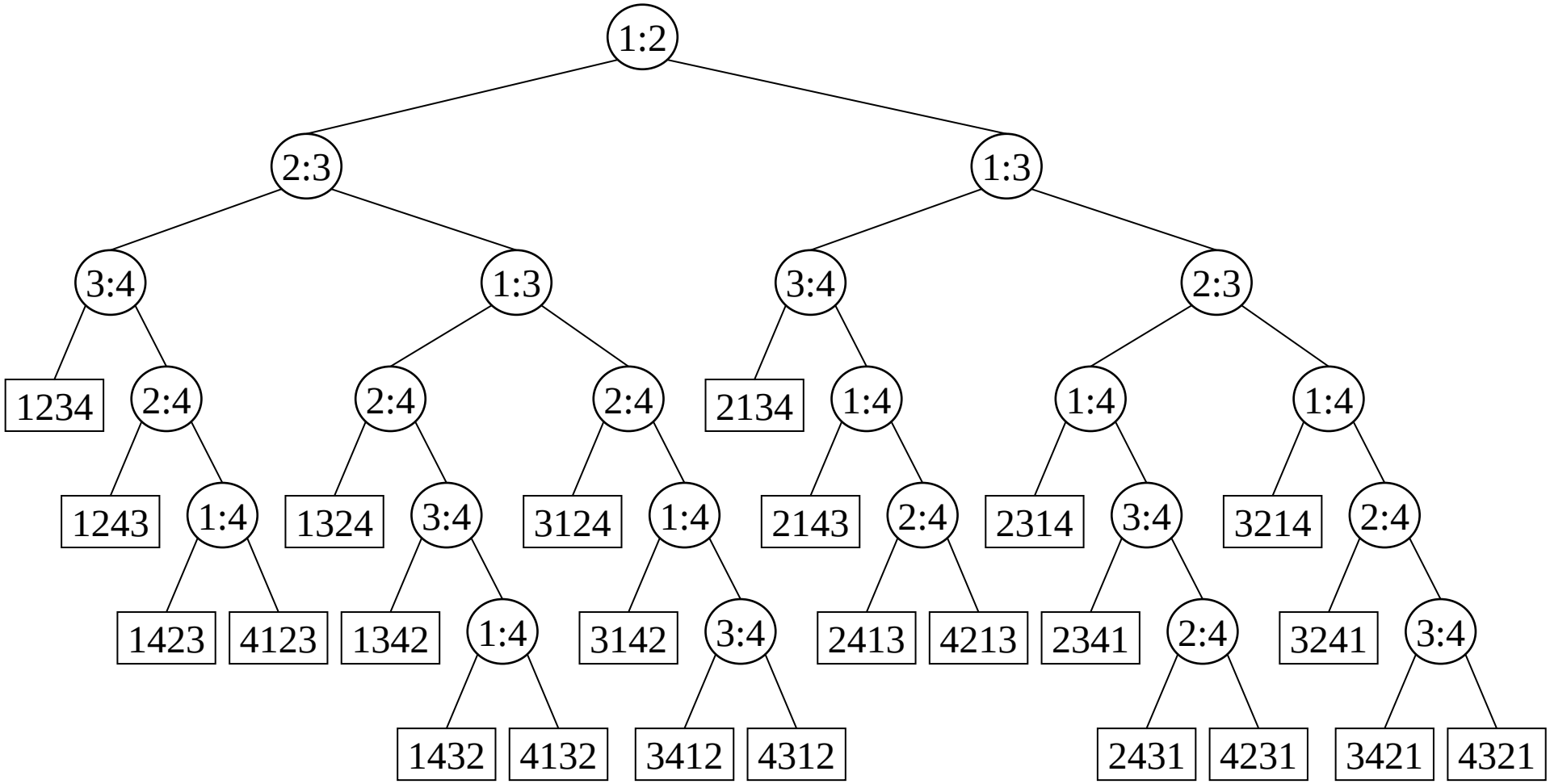
  entsprechen den von A durchgeführten Vergleichen, falls die Eingabe π_v ist

Beispiel:

Sortieren durch Einfügen für Folge

$F = <k_1, k_2, k_3, k_4>$ von 4 Schlüsseln

Beispiel eines Entscheidungsbaums



Vorsortierungsmaß, Inversionszahl

Spezielle Eingaben treten häufiger auf, Daten sind vorsortiert

Vorsortierungsmaße für eine Folge F

Anzahl der Inversionen

$$inv(F) = \left| \{(i,j) \mid 1 \le j < i \le n, \quad k_j > k_i\} \right|$$

Beispiel: 15 2 43 17 4 8 47

F aufsteigend sortiert: $inv(F) = 0$

F absteigend sortiert: $inv(F) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

inv mißt die globale Vorsortierung

Beispiel:

n/2+1,…,n,1,…,n/2

Anzahl aufsteigend sortierter Teilfolgen

$$runs(F) = \left| \{ i \mid 1 \le i \le n, \quad k_i > k_{i+1} \} \right| + 1$$

= Anzahl aufsteigend sortierter Teilfolgen

Beispiel: F: 15 2 43 17 4 8 47

F aufsteigend sortiert: $runs(F) = 1$

F absteigend sortiert: $runs(F) = n$

runs misst die lokale Vorsortierung

Längste aufsteigende Teilfolge

$las(F)$ = Länge der längsten aufsteigenden Subsequenz(F)

$$= \max\{ t \mid \exists \quad 1 \le i_1 < \cdots < i_t \le n, \quad k_{i_1} \le k_{i_2} \le \cdots \le k_{i_t} \}$$

$$1 \le las(F) \le n$$

Beispiel:

F: 15, 2, 43, 17, 4, 8, 47 $las(F) =$

$rem(F) = n - las(F)$

F aufsteigend: $rem(F) = 0$

absteigend: $rem(F) = n - 1$

Optimale Nutzung der Vorsortierung

Gegeben sei ein Vorsortierungsmaß m

Frage: Was ist ein m-optimales, allgemeines Sortierverfahren?

Ziel: Für jeden Wert m_0 von m soll nur die für Folgen dieses Vorsortiertheitsgrades nötige Schrittzahl verwendet werden.

Untere Schranke für Anzahl der nötigen Schlüsselvergleiche C_{m_0}?

Anzahl der Blätter im Entscheidungsbaum mit $m(F) \leq m_0$

$$\left| \{F \mid m(F) \leq m_0\} \right|$$

Optimale Nutzung der Vorsortierung

Definition:

Ein Sortierverfahren A heißt m-optimal, falls es eine Konstante c gibt, so dass für alle n und alle Folgen F mit Länge n die Zeit $T_A(F)$ zum Sortieren von F mit A wie folgt beschränkt ist:

$$T_A(F) \leq c \cdot (n + \log |\{F' \mid m(F') \leq m(F)\}|).$$

Adaptives Sortieren

• Adaptives Sortieren (A-Sort) ist eine Variante des Sortierens durch Einfügen.

• Man fügt jeweils nächsten Schlüssel in die bereits sortierte Teilfolge ein, indem man die Einfügestelle vom rechten Ende her sucht.

• Für eine Folge F = $<k_1, k_2, \ldots k_N>$ kann die Inversionszahl inv(F) wie folgt dargestellt werden:

$$inv(F) = |\{(i, j); 1 \leq i < j \leq N \text{ und } k_i > k_j\}| = h_1 + h_2 + \ldots + h_N$$

mit $h_j = |\{ i ; 1 \leq i < j \leq N \text{ und } k_i > k_j \}|$

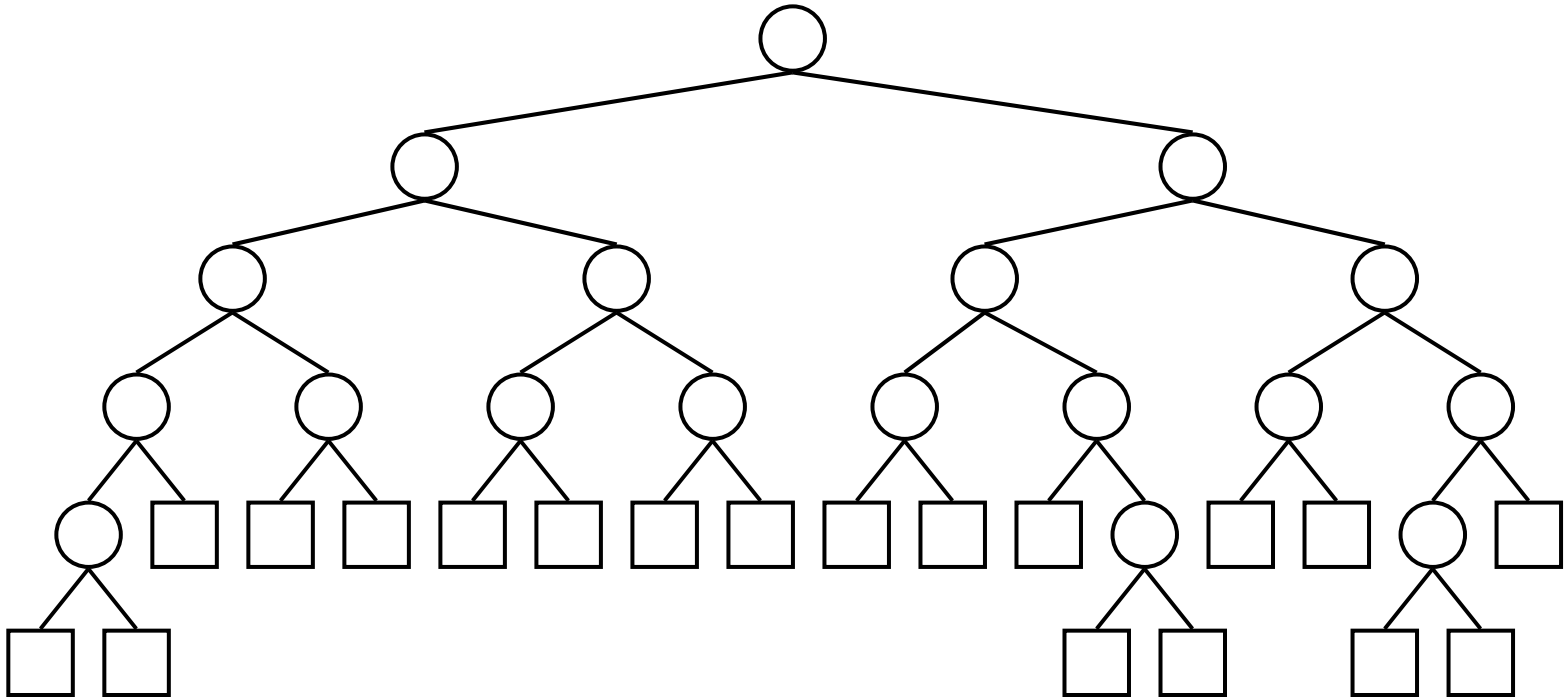
= Anzahl der dem j-ten Element k_j in der gegebenen Folge vorangehenden Elemente, die bei aufsteigender Sortierung k_j nachfolgen müssen

Beispiel

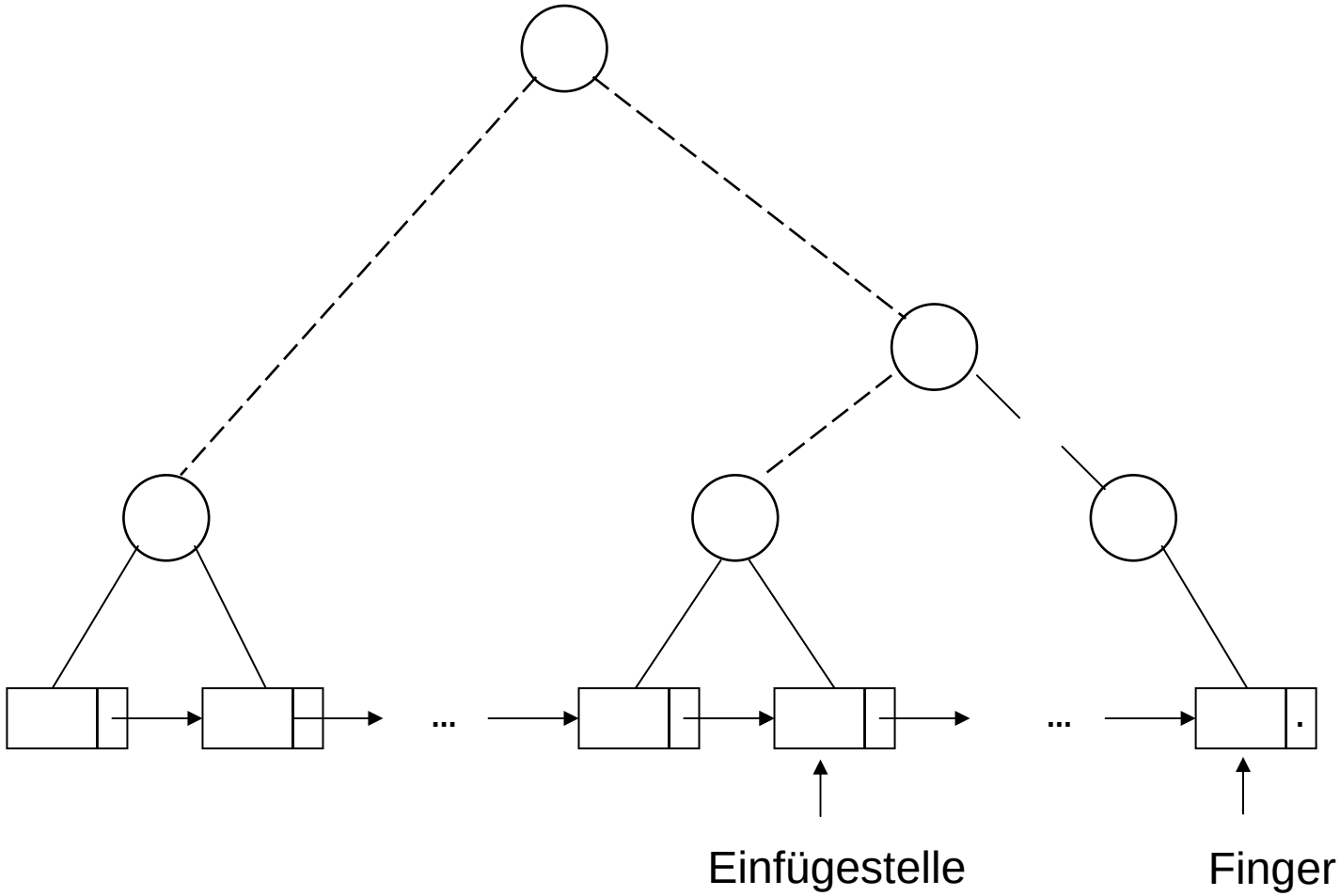
Sei F die Folge F = < 5, 1, 7, 4, 9, 2, 8, 3, 6 >

Nächstes einzufügendes Element k_i	h_i = Abstand der Einfügestelle vom Listenende	Nach Einfügen erhaltene Liste mit Markierung der Einfügestelle

Balancierter Blattsuchbaum



A-Sort Implementation



Balancierter Blattsuchbaum

- Verwende balancierten Blattsuchbaum zur Bestimmung der Einfügestelle vom rechten Ende der sortierten Teilfolge.
- Benutzt man beispielsweise AVL-Bäume oder Z-stratifizierte Bäume, so ist der gesamte Umstrukturierungsaufwand zum Einfügen von N Schlüsseln in den anfangs leeren Baum von der Größenordnung O(N).
- Der amortisierte Umstrukturierungsaufwand pro Einfügeoperation ist also konstant!
- Verwendung des balancierten Blattsuchbaumes erlaubt es, die Einfügestelle für das jeweils nächste Element in O(h + 1) Schritten zu bestimmen, wenn h der Abstand des Elementes vom Ende der bereits sortierten Teilliste ist.

Sortieraufwand

Aufwand zum Sortieren einer Folge F von N Elementen mit A-Sort:

T(F) = Umstrukturierungsaufwand + Suchaufwand

$= O(N) + \Sigma_{j=1}^{N} (\log (h_j + 1))$

$\Sigma_{j=1}^{N} (\log (h_j + 1))$

=

$= N \log (1 + inv(F)/N)$

Inv-Optimalität von A-Sort

- Falls inv(F) $\in$ O(N), ist *N log (1 + inv(F)/N) = O(N).*
- Falls inv(F) $\in$ O(N^2), ist *N log (1 + inv(F)/N) = O(N log N).*
- Man kann zeigen, dass gilt:

  $$log\ (|\{F' ; inv(F') \leq inv(F)\}|) \in \Omega\ (N\ log(1 + inv(F)/N))$$

  Daher ist A-Sort inv-optimal!

- A-Sort ist nicht runs-optimal, denn

  (1) dann müsste A-Sort alle Folgen mit 2 Runs in linearer Zeit sortieren!

  (2) Es gibt aber eine Folge mit nur zwei Runs, für die der Sortieraufwand mit A-Sort von der Ordnung Ω *(N log N) ist*

Folgen von N Elementen mit 2 Runs

Bew (1): Es gibt höchstens $O(2^N)$ Folgen mit nur 2 Runs:

Wähle alle 2^N möglichen Teilmengen von N Elementen, bilde jeweils aus einer Teilmenge den ersten Run und aus den restlichen Elementen den zweiten Run.

Wäre A-Sort runs-optimal, müsste für jede Folge F mit höchstens zwei Runs gelten:

$$T_A(F) \leq c\,(N + \log(\,|\{F' ; runs(F') \leq 2\}|\,)\,)$$

$$= c\,(N + \log 2^N\,)) = O(N)$$

Bew (2): Suche eine Folge mit 2 Runs, zu deren Sortierung A-Sort viel Zeit braucht!