

# Kap. 1: Priority-Queues

## 1.1 Fibonacci-Heaps



Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

2.-4. VO A&D WS 08/09 16./21./23. Oktober 2008

# Literatur für diese VO

- T. Ottmann und P. Widmayer: Algorithmen und Datenstrukturen, Spektrum Akademischer Verlag, 2002
- S. Krumke: Algorithmen und Datenstrukturen, Skript, TU Berlin, WS 2002/03
- K. Mehlhorn und P. Sanders: Algorithms and Data Structures: The Basix Toolbox, Springer, 2008
- Folienteil: ursprünglich von G. Klau, TU Wien, SS 2004 (jetzt: FU Berlin)

# Überblick

Einführung Fibonacci

Einführung Fibonacci-Heaps

Realisierung

Amortisierte Analyse

# Leonardo von Pisa



- Fibonacci = „Sohn des Bonacci“
- ca. 1170 – 1240, Pisa
- Mathematiker, Gleichungslehre
- Liber Abaci 1202

$$F(n+2)=F(n)+F(n+1)$$

$$F(1) = 1$$

$$F(2) = 1$$

- Berühmte Kaninchenaufgabe:

*„A certain man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the second month on becomes productive?“*

$n$	1	2	3	4	5	6	7	8	9	10	11	12
$F(n)$	1	1	2	3	5	8	13	21	34	55	89	144
$F(n)/F(n-1)$		1,0	2,0	1,5	1,667	1,60	1,625	1,615	1,619	1,618	1,618	1,618
$F(n-1)/F(n)$		1,0	0,5	0,667	0,600	0,625	0,615	0,619	0,618	0,618	0,618	0,618

# Leonardo von Pisa

- Fibonacci = „Sohn des Bonacci“
- ca. 1170 – 1240, Pisa
- Mathematiker, Gleichungslehre
- Liber Abaci 1202



$$F(n+2) = F(n) + F(n+1)$$

$$F(1) = 1$$

$$F(2) = 1$$

- $\Phi = (1 + \sqrt{5})/2 = 1,6180339887\dots$
- $\underline{\Phi} = (1 - \sqrt{5})/2 = -0,6180339887\dots$  (Konjugierte)
- $\Phi^{-1} = 2/(1 + \sqrt{5}) = -\underline{\Phi} = (\sqrt{5} - 1)/2 = 0,6180339887\dots$
- $\Phi^{-1}$ : „Der Goldene Schnitt“

n	1	2	3	4	5	6	7	8	9	10	11	12
F(n)	1	1	2	3	5	8	13	21	34	55	89	144
F(n)/ F(n-1)		1,0	2,0	1,5	1,667	1,60	1,625	1,615	1,619	1,618	1,618	1,618
F(n-1)/ F(n)		1,0	0,5	0,667	0,600	0,625	0,615	0,619	0,618	0,618	0,618	0,618

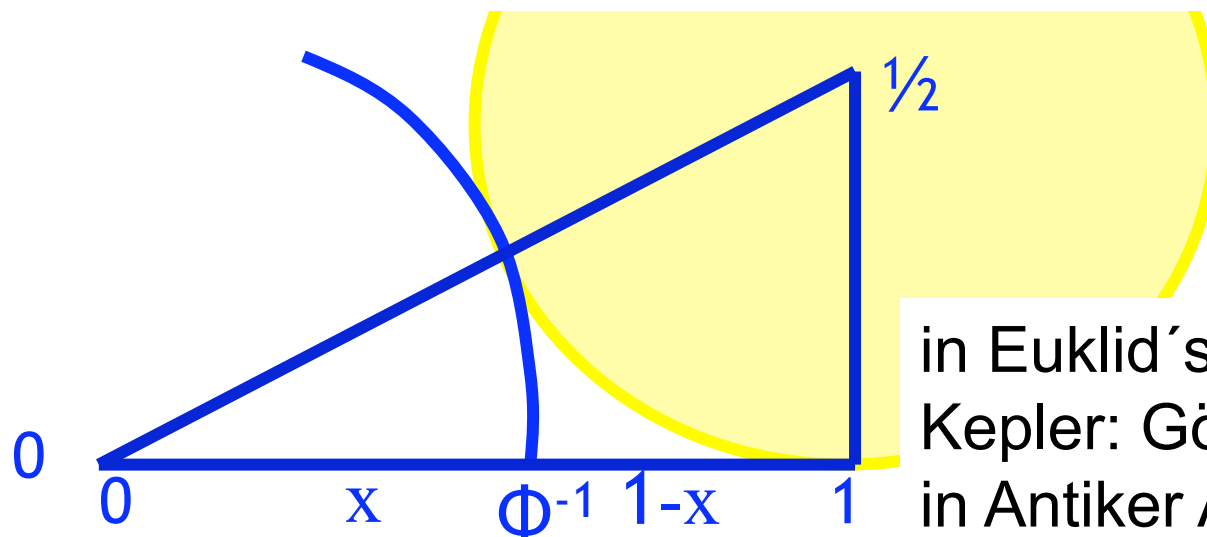
# Der Goldene Schnitt

$$\Phi^{-1} = 2/(1+\sqrt{5}) = (\sqrt{5}-1)/2 = 0,6180339887\dots$$

$\Phi^{-1}$  ist bekannt als der goldene Schnitt.

Er ergibt sich durch die Zerlegung einer Strecke  $a$  in zwei positive Summanden  $x$  und  $a-x$ , so dass  $x$  geometrisches Mittel von  $a$  und  $a-x$  ist, d.h.

$x^2 = a(a-x)$ . Es gilt:  $x/a = (a-x)/x$ .



in Euklid's Elementen &  
Kepler: Göttliche Teilung  
in Antiker Architektur

# Prioritätswarteschlangen

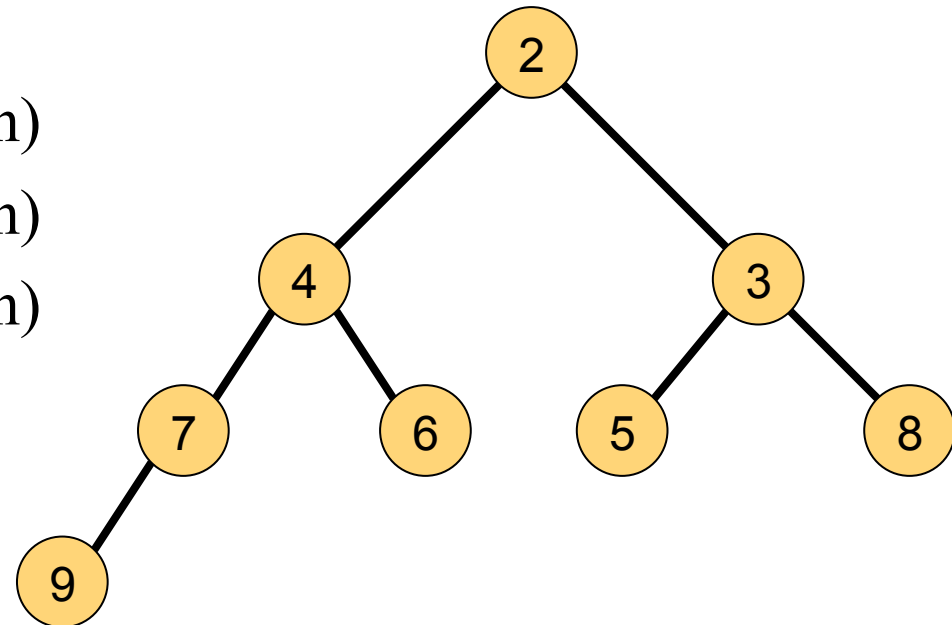
- Ausgangsfrage: Wie kann man den Datentyp Prioritätswarteschlange realisieren?
  - Anwendungen, z.B.
    - kürzeste Wege mit Dijkstra
    - MST mit Prim
    - Finden eines minimalen Schnittes

**Operationen:**  
**Insert**  
**ExtractMin**  
**DecreaseKey**

# Binäre Heaps

Heapeigenschaft: Elter ist nicht größer als seine Kinder

- **Insert:**  $O(\log n)$
- **ExtractMin:**  $O(\log n)$
- **DecreaseKey:**  $O(\log n)$



n Knoten



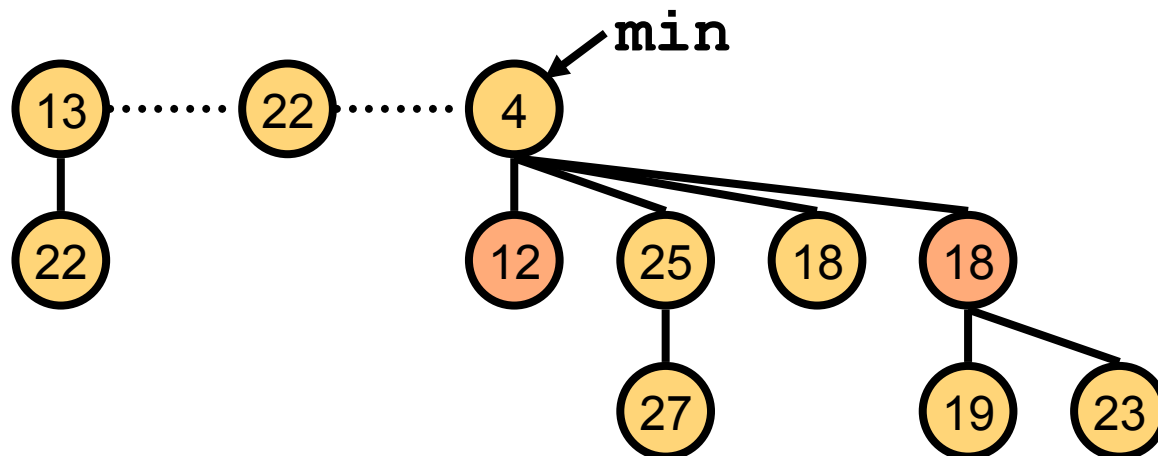
# Fibonacci-Heaps (Fredman & Tarjan 1987)

Operation	Bin. Heap	Fibonacci-Heap
<b>Insert</b>	$O(\log n)$	$O(1)$
<b>ExtractMin</b>	$O(\log n)$	$O(\log n)^*$
<b>DecreaseKey</b>	$O(\log n)$	$O(1)^*$

\* Nicht ganz fair, weil amortisierte Analyse

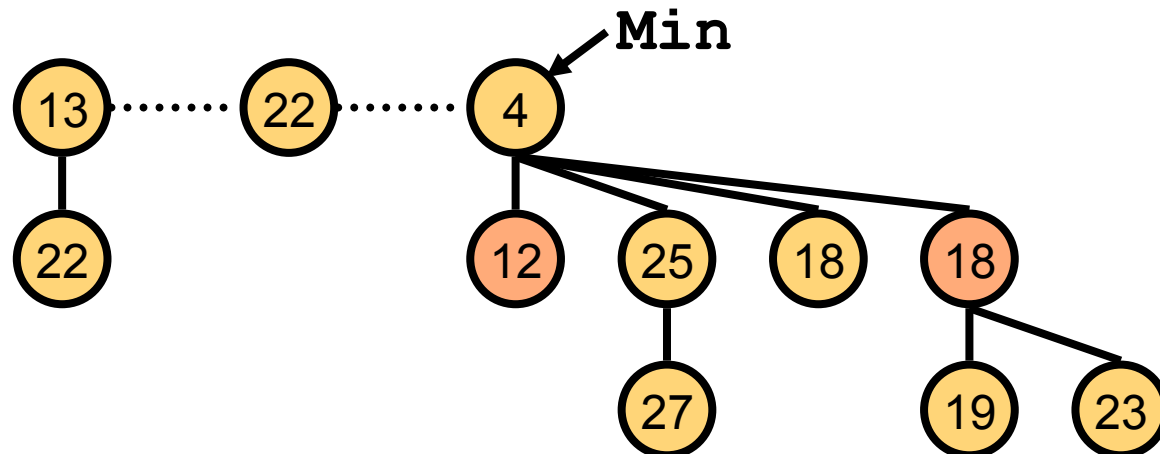
# Fibonacci-Heaps: Features

- Menge von Bäumen, die alle die Heapeigenschaft haben
- Globaler **Min**-Pointer
- Bäume nicht zwangsläufig binär
- Jeder Knoten ist entweder markiert oder nicht



# Fibonacci-Heaps: Realisierung

- Alle Wurzeln durch doppelt-verkettete, zyklisch geschlossene Liste verbunden: Wurzelliste
- Jeder Knoten hat Zeiger auf seinen Elter und auf einen seiner Kinder (**Elter**, **Kind**)
- Alle Kinder sind untereinander doppelt zyklisch verkettet
- Jeder Knoten hat Felder  
**Schlüssel**, **Mark**, **Rang** (Anzahl der Kinder)



# Verschmelze ( $h_1, h_2$ )

- Verschmelzen zweier F-Heaps  $h_1$  und  $h_2$ :
  - Hänge beide Wurzellisten aneinander
  - Der **Min**-Pointer zeigt zum kleineren der beiden Min-Zeiger
- Aufwand:  $O(1)$  Zeit.

# Insert (v)

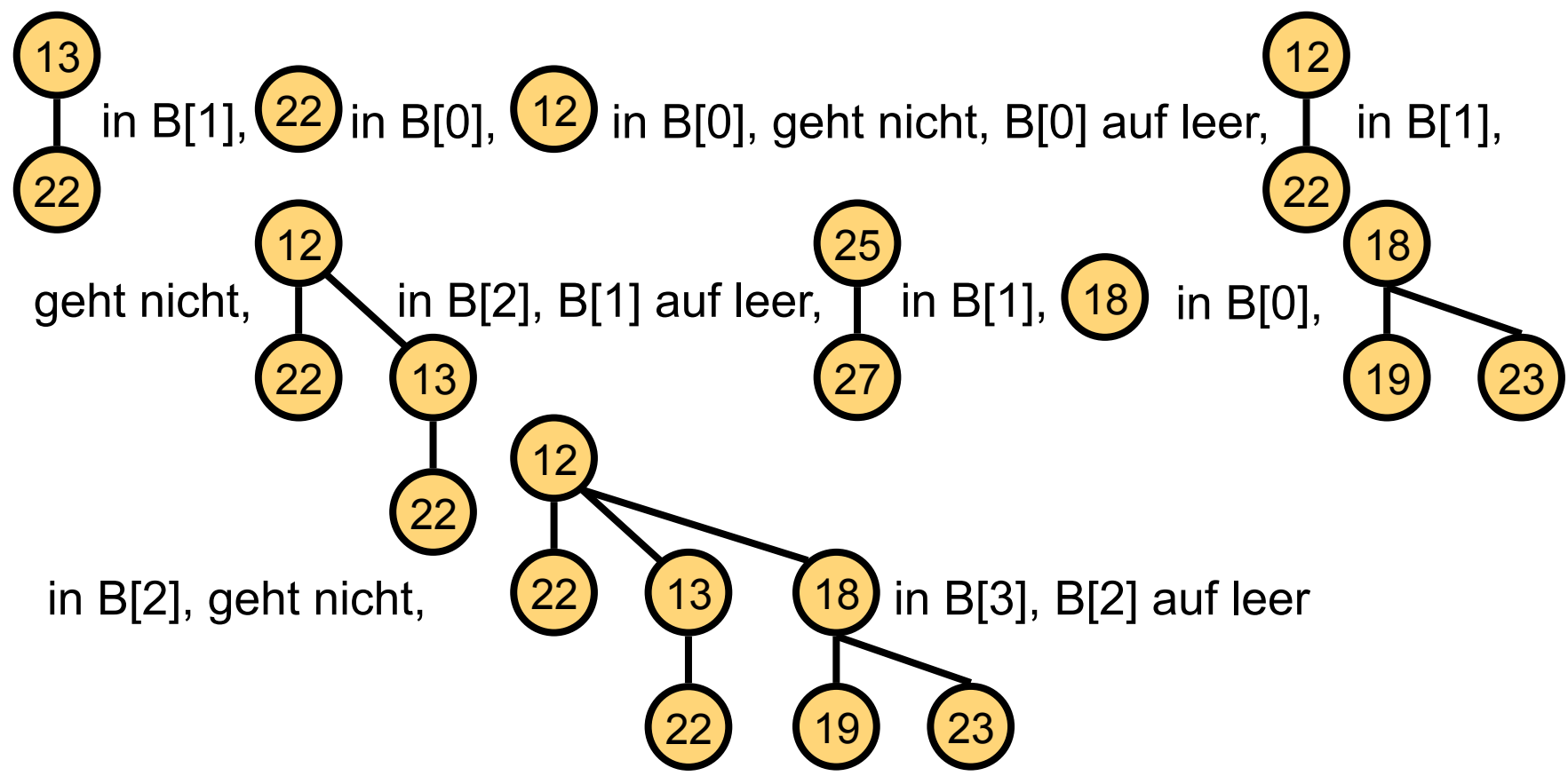
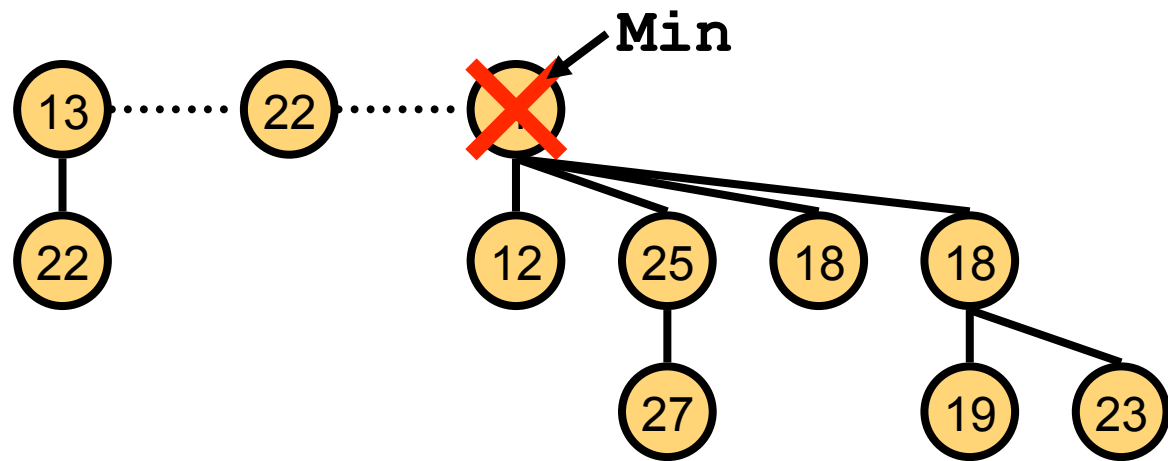
- Insert(v) in einen F-Heap h:
  - Bilde einen F-Heap  $h'$  aus einem einzigen Knoten v (**Mark**=0, **Rang**=0)
  - Verschmelze h und  $h'$  zu einem neuen F-Heap
- Aufwand:  $O(1)$  Zeit. Die Faulheit rächt sich aber bei **ExtractMin...**

# Fibonacci-Heaps: ExtractMin

- Es ist einfach, das Minimum zu finden (**Min-Pointer**)
- Entferne den Minimalknoten aus der Wurzelliste von  $h$
- Hänge alle Kinder von Min in die Wurzelliste ein
- Durchlaufe alle Wurzeln, um neues Minimum zu finden
- **Idee:** Wenn man das schon machen muss, kann man dabei auch aufräumen
  - Wir sorgen dafür, dass keine zwei Wurzeln den gleichen Grad haben
  - Dazu: Feld  $B[0..n-1]$  mit der Interpretation  $B[i] = v \leftrightarrow \deg(v) = i$

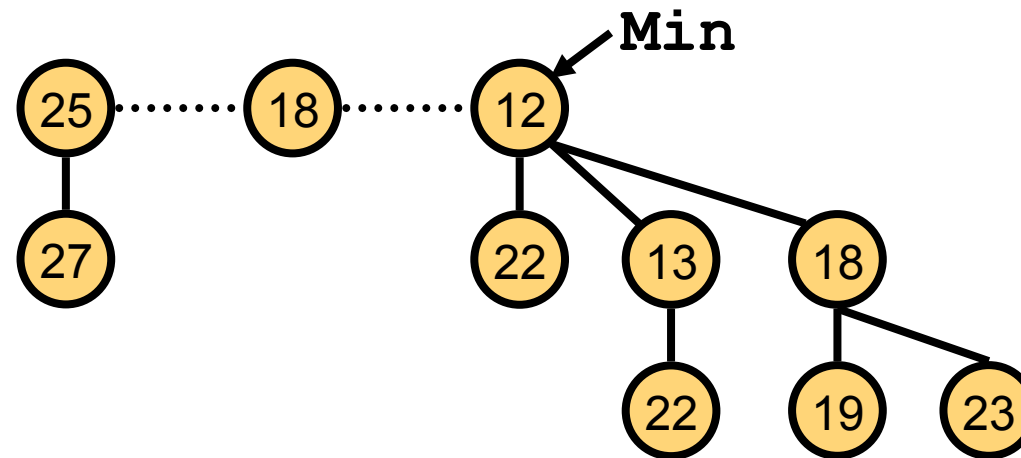
# Fibonacci-Heaps: ExtractMin

- Für alle Wurzeln  $v$ :
  - Wiederhole
    - $d = \text{deg}(v)$
    - falls  $B[d]$  leer setze  $B[d] = v$
    - sonst: mache größere Wurzel zum Kind der kleineren;
      - $v = \text{kleinere Wurzel}$
      - $B[d+1] = v$  // der Grad von  $v$  ist jetzt  $d + 1$
      - $B[d] = \text{leer}$  // wieder frei
  - bis ein leeres Feld gefunden wurde





# Fibonacci-Heaps: ExtractMin



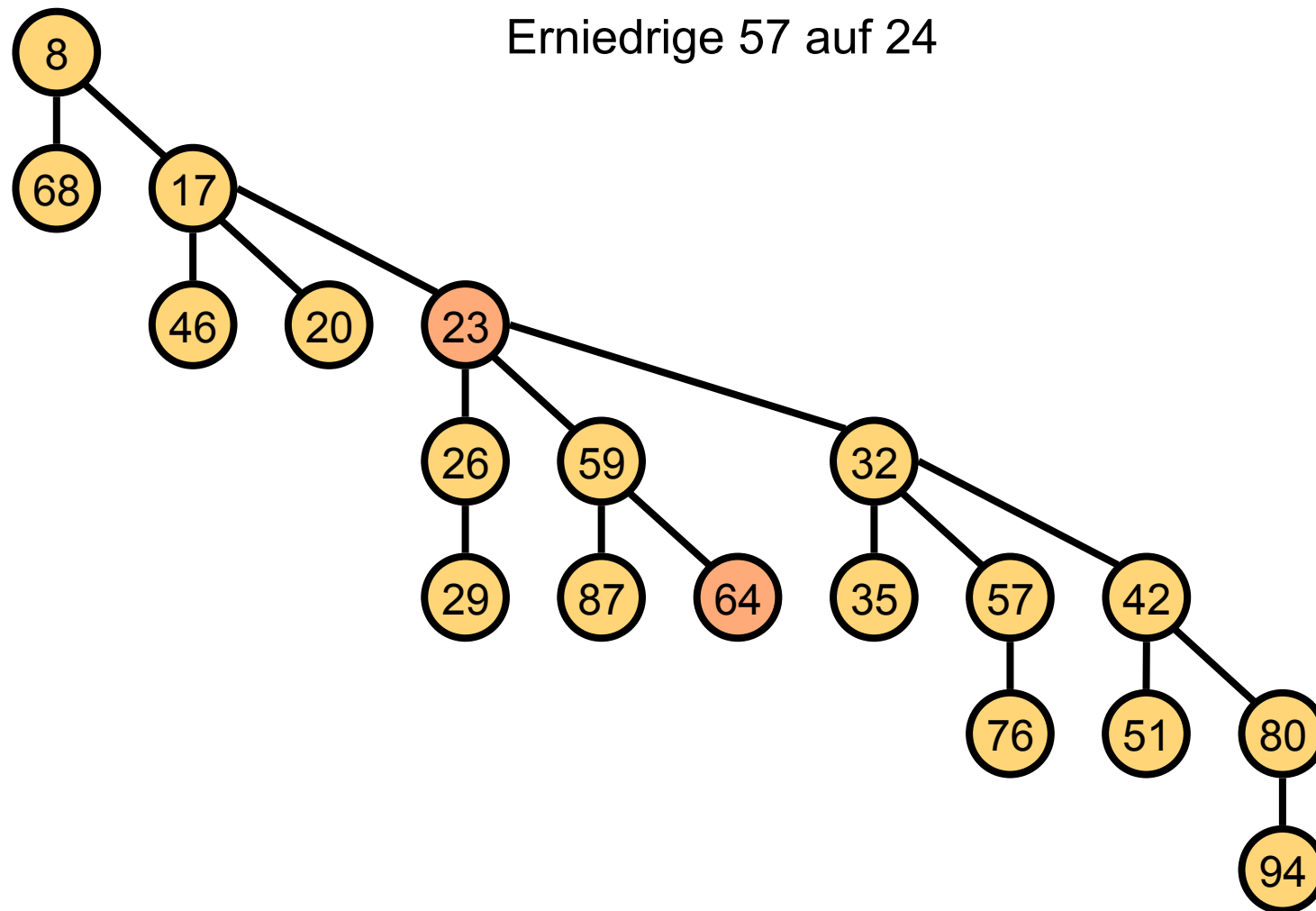
- Laufzeit:  $O(d + W_1 + m) = O(d + W_1)$ ,
  - wobei  $d = \text{deg}(\text{gelöschte Wurzel})$
  - und  $W_1 = \#\text{Wurzeln vor dem Löschen}$
  - und  $m = \text{Anzahl der merge Operationen (Aufräumen)}$
- denn: höchstens 1 merge-Aufruf pro Wurzel
- genauere Analyse: später

# Fibonacci-Heaps: DecreaseKey

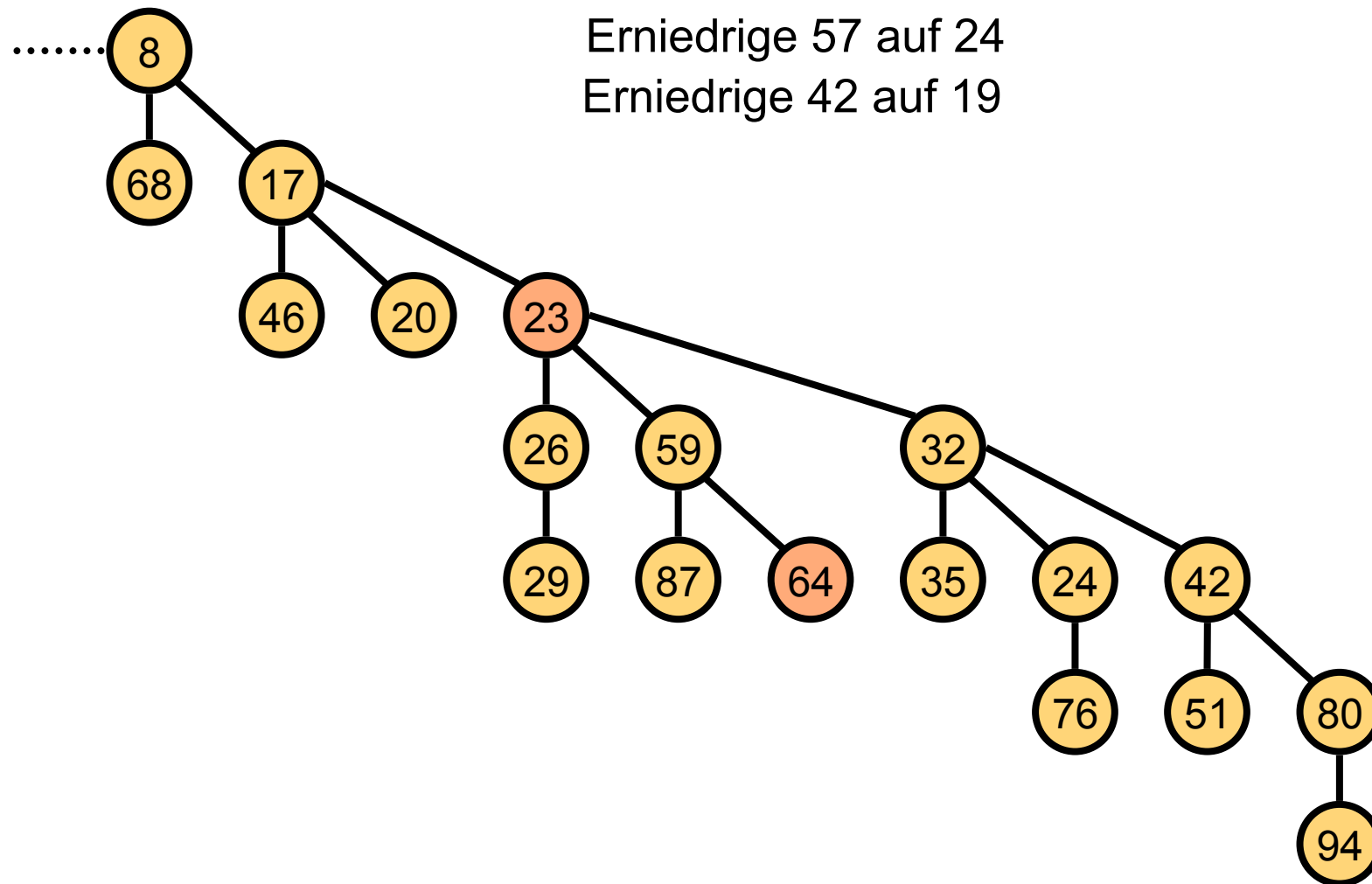
- **DecreaseKey** ( $v$ , newKey) :
  - Reduziere Schlüssel von  $v$  auf newKey;
  - Wiederhole:
    - Trenne  $v$  von  $p = \mathbf{Elter}(v)$  ab;
    - Füge  $v$  in Wurzelliste ein; setze  $\mathbf{Mark}(v) = 0$ ;
    - Reduziere **Rang** ( $p$ )
    - Setze  $v = p$ ;
  - solange  $\mathbf{Mark}(p) = 1$ ; // trenne solange **Elter**( $v$ ) ab,  
// bis unmarkiert
  - Setze  $\mathbf{Mark}(p) = 1$ ;

Markierung verhindert, dass F-Heaps zu dünn werden

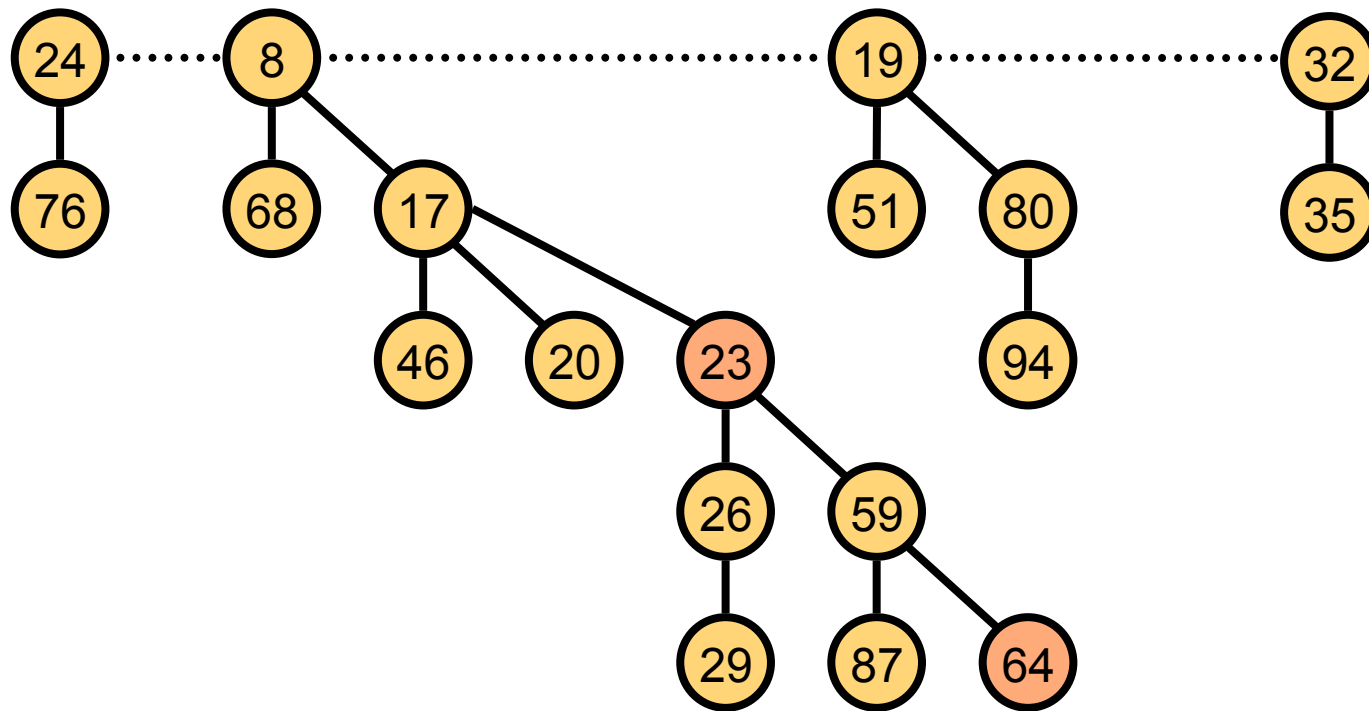
# Fibonacci-Heaps: DecreaseKey



# Fibonacci-Heaps: DecreaseKey



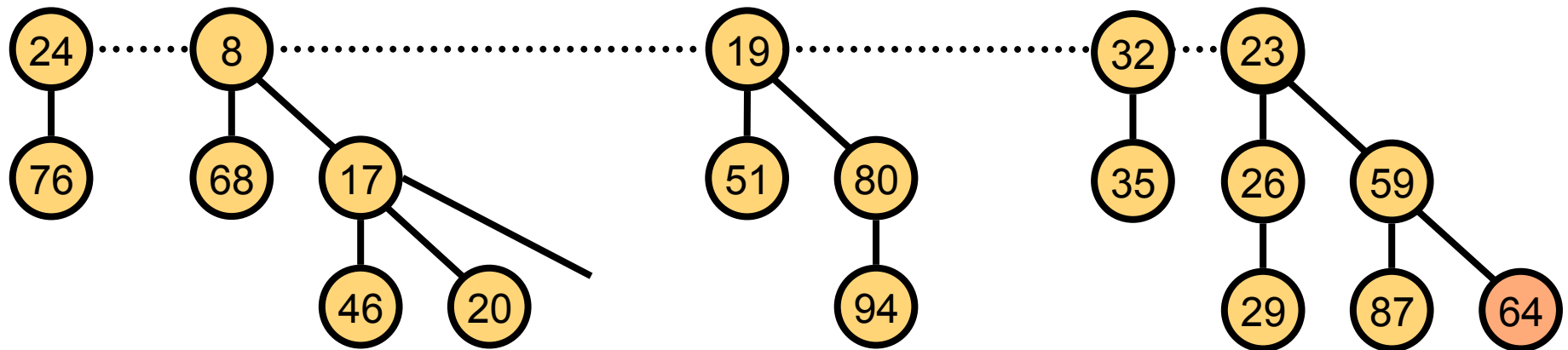
# Fibonacci-Heaps: DecreaseKey



Erniedrige 42 auf 19

noch nicht fertig:

# Fibonacci-Heaps: DecreaseKey



Erniedrige 42 auf 19

# Fibonacci-Heaps: Analyse

Wir wollen jetzt zeigen, dass die Knotengrade im F-Heap durch  $O(\log n)$  beschränkt sind (deswegen auch die Markierung).

**Lemma F1:** Sei  $p$  ein Knoten vom Grad  $d$  und seien  $v_1, \dots, v_d$  die Kinder von  $p$  in der zeitlichen Reihenfolge, in der sie an  $p$  (durch Verschmelzen) angehängt wurden, dann gilt: Das  $i$ -te Kind von  $p$  hat mindestens Grad  $i-2$ .

**Beweis:** Als das  $i$ -te Kind  $v_i$  an  $p$  angehängt wurde, mussten  $p$  und  $v_i$  mindestens Grad  $i-1$  gehabt haben und beide denselben Grad. Seitdem kann  $v_i$  höchstens 1 Kind verloren haben; es besitzt also mindestens noch Grad  $i-2$ .

# Fibonacci-Heaps: Analyse

**Lemma F2:** Jeder Knoten  $p$  vom Grad  $k$  eines F-Heaps ist Wurzel eines Teilbaumes mit mindestens  $F_{k+2}$  Knoten.

**Bemerkung:**  $F_{k+2}$  ist die  $k+2$ -te Fibonaccizahl. Deswegen haben Fredman und Tarjan den Namen Fibonacci-Heap eingeführt.



# Beweis von Lemma F2

Beweis: Sei  $p$  ein Knoten vom Grad  $k$  und seien  $v_1, \dots, v_k$  die Kinder von  $p$  in der zeitlichen Reihenfolge, in der sie an  $p$  (durch Verschmelzen) angehängt wurden.

Sei  $S_k$  die Minimalzahl von allen Nachfolgern eines Knotens  $p$  vom Grad  $k$  (einschließlich  $p$ )

z.z.:  $S_k \geq F_{k+2}$

$v_1$  hatte mind. Grad 0: Beitrag zu  $S_k$ :  $S_0 \geq 1$  ✓

$v_2$  hatte ursprünglich mind. Grad 1, und jetzt mindestens noch Grad 0:  $S_0 \geq 1$  ✓

$v_3$  hatte ursprünglich mind. Grad 2, und jetzt mindestens noch Grad 1:  $S_1 \geq 2$  ✓

$v_k$  hatte ursprünglich mind. Grad  $k-1$ , und jetzt mindestens noch Grad  $k-2$ :  $S_{k-2}$  ✓ (s. Lemma F1)

$$\Rightarrow S_k \geq \underbrace{1}_{\text{von } p} + \underbrace{S_0}_{\text{von } v_1} + S_0 + S_1 + \dots + S_{k-2} \geq 1 + 1 + \sum_{i=0}^{k-2} S_i \text{ für } k \geq 2$$

# Beweis von Lemma F2 ff

Beweis ff: Soweit gezeigt:  $S_k \geq 1 + 1 + \sum_{i=0}^{k-2} S_i$

Aus Def. der Fibonacci-Zahlen folgt:  $F_{k+2} = 1 + 1 + \sum_{i=2}^k F_i$

Beweis: Wir zeigen für  $k \geq 0$  gilt:  $F_{k+2} = 1 + \sum_{i=0}^k F_i$

$$k=0: F_2 = 1 + F_0 = 1 + 0 = 1$$

Ann.: für alle bis  $k+1$  schon bewiesen, z.z. für  $k+2$ :

$$F_{k+2} = F_k + F_{k+1} = F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right) = 1 + \sum_{i=0}^k F_i$$

# Beweis von Lemma F2 fff

Beweis fff: Soweit gezeigt:  $S_k \geq 1 + 1 + \sum_{i=0}^{k-2} S_i$

Aus Def. der Fibonacci-Zahlen folgt:  $F_{k+2} = 1 + 1 + \sum_{i=2}^k F_i$

z.z. bleibt:  $S_k \geq F_{k+2}$

k=0:  $S_0=1 \geq F_2=1$  ✓

k=1:  $S_1 \geq 2 \geq F_3=2$  ✓

k≥2: Ann.  $S_i \geq F_{i+2}$  gilt für  $i \leq k-1$ :

$$S_k \geq 2 + \sum_{i=0}^{k-2} S_i \geq 2 + \sum_{i=0}^{k-2} F_{i+2} = 2 + \sum_{i=2}^k F_i = F_{k+2} \quad \checkmark$$

# Fibonacci-Heaps: Analyse

Also jetzt gezeigt:

**Lemma F2:** Jeder Knoten  $p$  vom Grad  $k$  eines F-Heaps ist Wurzel eines Teilbaumes mit mindestens  $F_{k+2}$  Knoten.

**Lemma F3:** Jeder Knoten eines F-Heaps mit insgesamt  $N$  Knoten hat einen Grad von höchstens  $k \leq \log_{\phi} N$ .

Beweis: Es gilt  $F_{k+1} \geq \phi^k$  (Goldene Schnitt) (o.Bw.)

$$N \geq S_k \geq F_{k+2} \geq \phi^{k+1}$$

$$\rightarrow k+1 \leq \log_{\phi} N \quad \rightarrow \quad k \leq \log_{\phi} N$$

**Folgerung:** Bei **ExtractMin** wird die Wurzelliste um höchstens  $O(\log N)$  Wurzeln verlängert.

# Fibonacci-Heaps: Analyse

- Oops:  $n$  mal **insert**, 1 **ExtractMin**  $\rightarrow \Theta(n)$ 
  - Worst case. Schlechter als binärer Heap!
  - Aber selten so schlecht.
  - Intuition: **Insert** baut Spannung/Potenzial auf, das **ExtractMin** verlangsamt, sich aber dabei abbaut. Nachfolgende **ExtractMin** haben es dafür leichter...
  - Potenzial baut sich nur langsam auf (über  $n$  Einfügeoperationen)  $\rightarrow$  durchschnittliche Kosten über alle Operationen  $O(1)$ .
- Wir brauchen eine amortisierte Analyse.
  - Nichts für eine Realzeitumgebung.

# Fibonacci-Heaps: Amortisierte Analyse

- Führt man eine beliebige Folge von Operationen aus, dann ist die dafür insgesamt benötigte Zeit beschränkt durch die gesamte amortisierte Zeit.
- Die amortisierte Zeit einer einzelnen **ExtractMin** Operation ist  $O(\log N)$ , die amortisierte Zeit aller anderen Operationen ist  $O(1)$ .

# Amortisierte Analyse via Potenzialmethode

- Dazu Potenzial mathematisch beschreiben:
  - Funktion  $\Phi$ , die jeden Heap auf eine reelle Zahl abbildet
- **Amortisierte Kosten** einer Operation sind
  - tatsächliche Kosten +  $\Delta\Phi$
  - also auch Belastung durch zukünftige Kosten (**Insert**) oder Entschärfung von teuren Operationen ( $\Delta\Phi$  negativ).
- Sei  $\mathbf{o}_1, \dots, \mathbf{o}_n$  eine Folge von  $n$  Operationen
  - $\mathbf{a}_i$  amortisierte Kosten für Operation  $\mathbf{o}_i$
  - $\mathbf{t}_i$  tatsächliche Kosten für Operation  $\mathbf{o}_i$
  - $\Phi_i$  Potenzial direkt nach Operation  $\mathbf{o}_i$
  - $\Phi_0$  Potenzial vor  $\mathbf{o}_1$

# Amortisierte Analyse: Vorgehen

- Dann ist die Summe der amortisierten Kosten

$$\sum_{i=1}^n a_i = \sum_{i=1}^n (t_i + \Phi_i - \Phi_{i-1}) = \Phi_n - \Phi_0 + \sum_{i=1}^n t_i$$

- Wir wählen eine Potenzialfunktion, die
  - nichtnegativ und
  - am Anfang gleich Null ist
- Dann ist  $\Phi_n - \Phi_0 \geq 0$  und damit  $\sum t_i \leq \sum a_i$



# Amortisierte Analyse: Wahl von $\phi$

- Konkrete Wahl von  $\Phi$ ?
  - Was kann später große Kosten verursachen?
  - Sicher Anzahl der Wurzeln  $W$ !
  - Wir wählen provisorisch erst einmal:  $\Phi = \alpha W$
  - *(wird gleich noch erweitert werden...)*

# Amortisierte Analyse: ExtractMin

- Entfernter Knoten hatte Grad  $d$ , vorher hatte der F-Heap  $W_1$  Wurzeln, nach **ExtractMin**  $W_2$  Wurzeln, dabei  $m$  *merge* Operationen beim Aufräumen
- **Es gilt:**  $W_2 = W_1 + d - m$  und  $m \leq W_1 + d$  (*denn bei jedem merge verschwindet eine Wurzel*)
- Tatsächliche Kosten  $t_i = c(W_1 + d)$ ,  $c$  Konst.
- $\Delta\Phi = \alpha(W_2 - W_1)$ , also amortisierte Kosten:  
 $a_i \leq c(W_1 + d) + \alpha(W_2 - W_1) = (c - \alpha)W_1 + cd + \alpha W_2$
- Wir wählen  $\alpha = c$ , also  $a_i = cd + \alpha W_2$ 
  - aber  $d = O(\log n)$  und  $W_2 = O(\log n)$ , denn alle Wurzeln haben unterschiedlichen Grad
- also  $a_i = O(\log n)$

# Amortisierte Analyse: DecreaseKey

- schafft  $k$  neue Wurzeln
  - tatsächliche Kosten  $t_i = c'(k + 1)$ ,  $c'$  Konstante
- **Problem:** Dies ist teuer und erhöht Potenzial...
- **Lösung:** Zweite Quelle für  $\Phi$ : markierten Knoten  $A$ 
  - Werden leicht zu Wurzeln und verursachen dabei noch Arbeit, also  $\Phi = \alpha W + \beta A$
- $\Delta\Phi \leq \alpha k + \beta(-k+1+1) = \alpha k + \beta(2 - k)$ , denn alle neuen Wurzeln waren markiert (bis auf vielleicht eine), höchstens ein Knoten wird neu markiert
- Amortisierte Kosten von **DecreaseKey**  
 $a_i \leq c'(k + 1) + \alpha k + \beta(2 - k) = (c' + \alpha - \beta)k + c' + 2\beta$
- Wähle  $\beta = c' + \alpha$ , es folgt  $a_i = 3c' + 2\alpha = O(1)$

# Amortisierte Analyse: Insert

- Müssen wir **ExtractMin** neu abschätzen?
  - Nein, denn dort verlieren höchstens Knoten ihre Markierung
- **Insert**:
  - Erhöht die Anzahl der Wurzeln um 1
  - $a_i = t_i + \alpha = O(1 + \alpha) = O(1)$
- Klar, dass  $\Phi$  nichtnegativ und am Anfang 0

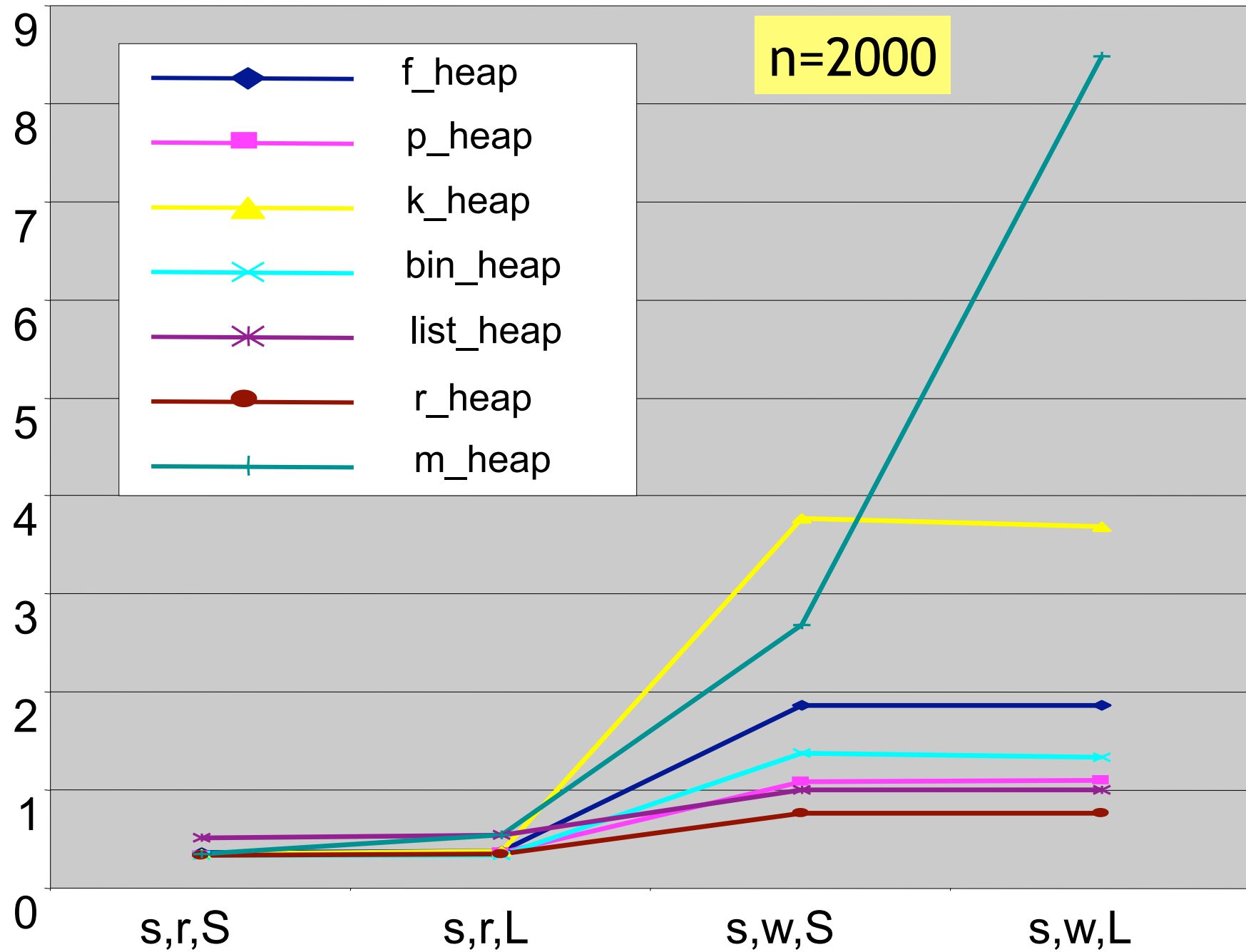
Beweis-Ende

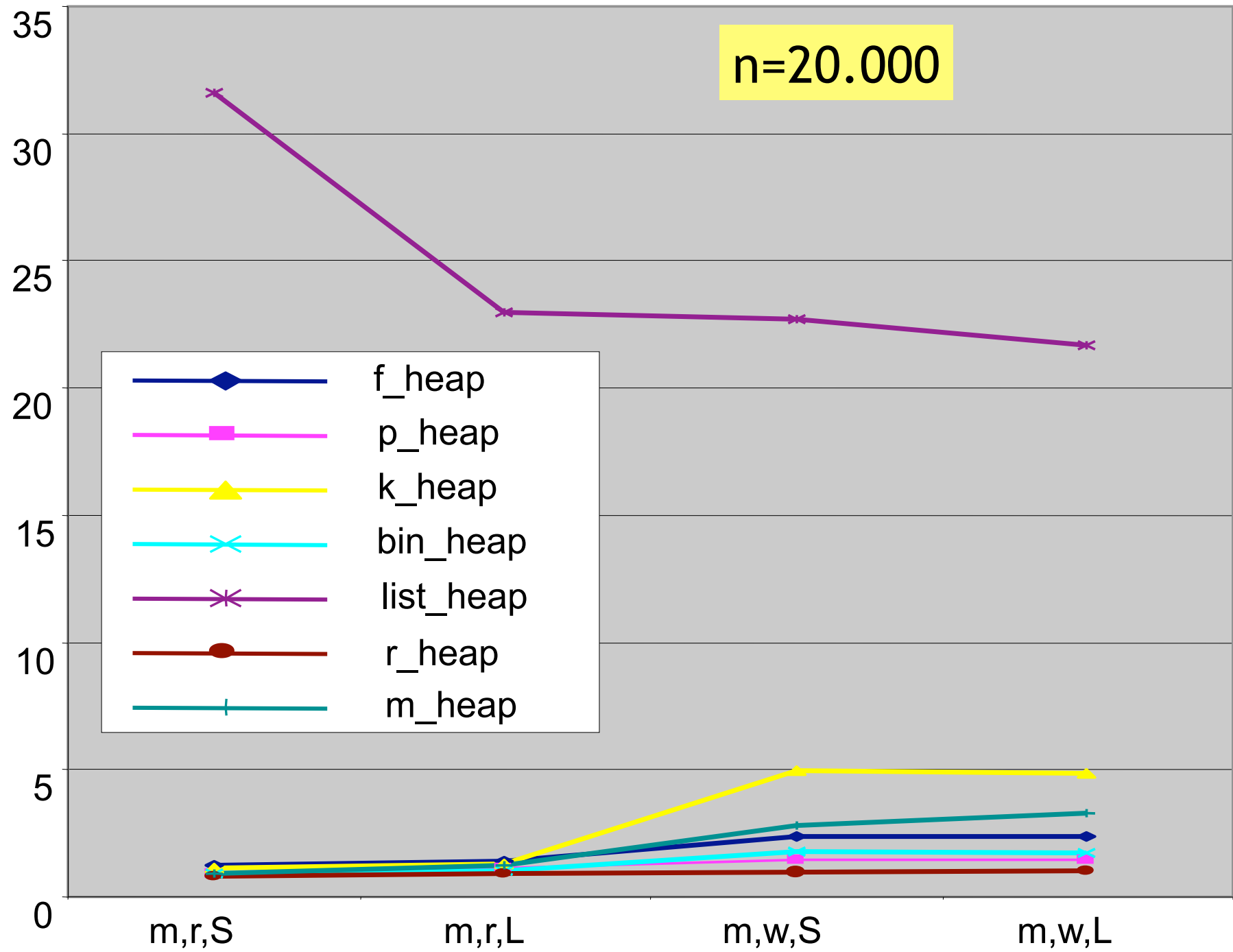
# Experimenteller Vergleich

- f\_heap: Fibonacci-Heaps
- p\_heap: Pairing Heaps ähnlich wie F-Heap, Rebalancierung durch paarweises merge bei **ExtractMin**
- k\_heap: k-ary Heaps
- bin\_heap: Binary Heaps
- list\_pq: lineare unsortierte Liste
- r\_heap: Redistributive Heaps (AMOT90)\*
- m\_heap: Monotone Heaps\*
  
- \*: nur monotone Heaps
- die letzten drei Varianten nur Integer in best. Bereich mehr: s. Übung

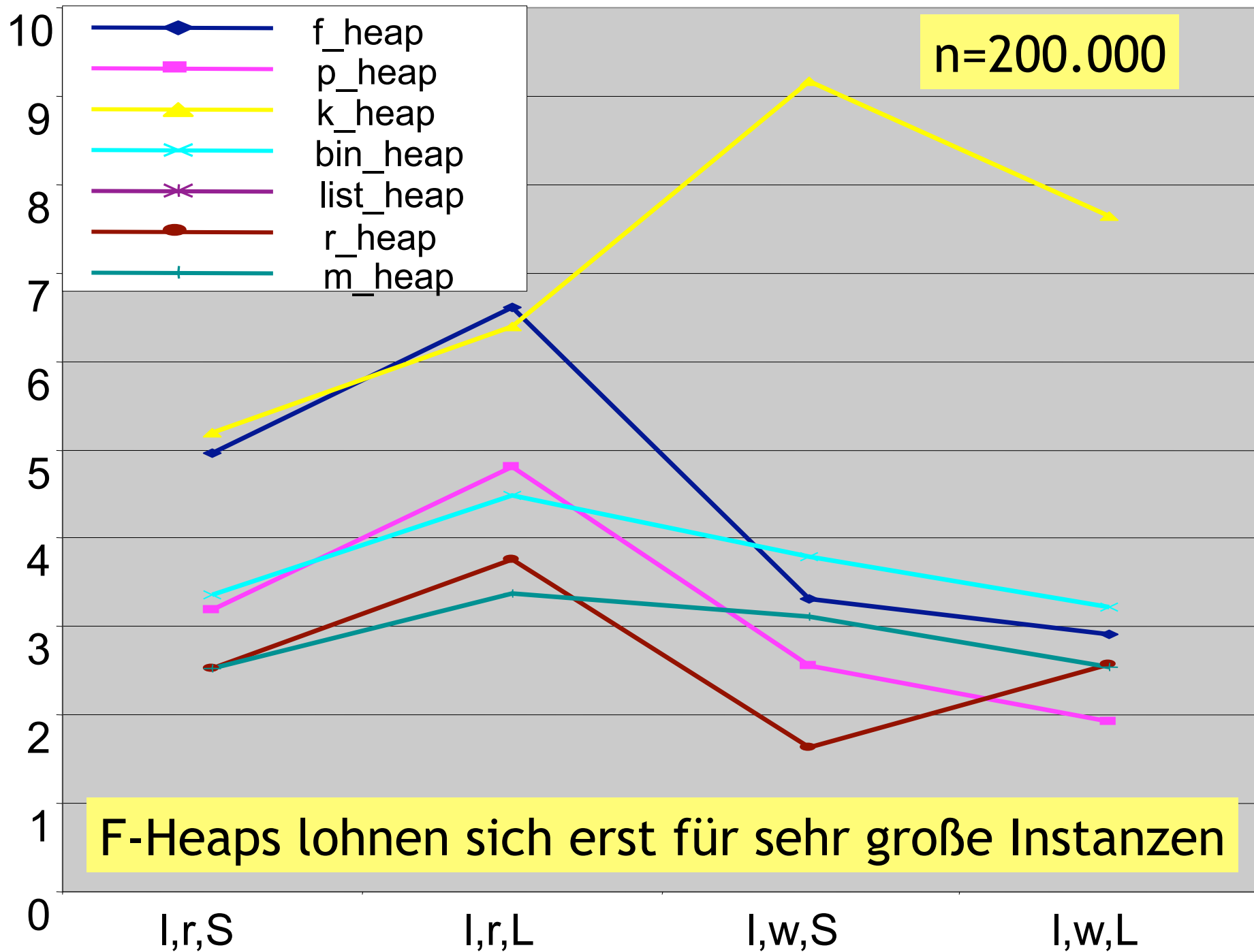
# Experimenteller Vergleich

- Mehlhorn & Näher: LEDA Buch 2000
- Vergleich innerhalb von Dijkstra
- Je 1 Graph pro Datenpunkt (!)
- Knotenzahl mit je 2000 (**s**), 20.000 (**m**) und 200.000 (**l**)
- je 4 verschiedene Instanzen für **s**, **m** und **l**:
- Kantengewichte random (**r**) aus  $[0..M-1]$  für  $M=100$  (**S**) und  $M=1000$  (**L**)
- Worst-Case Kantengewichte (**w**), so dass die Update-Zeit maximal wird ( $c=0$  (**S**) und  $c=10.000$  (**L**))
- Gute Erfolge werden mit Bucket-Heaps und HOT-Queues erzielt (s. experimentelle Studie Übung)









# Ende 1.1 Fib.-Heap