Fibonacci-Heaps

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering LS11

Fakultät für Informatik, TU Dortmund

Selected Topics on Combinatorial Optimization

Vienna Graduate School on Computational Optimization

Lecture 9, November 15, 2018

Literature

- T. Ottmann and P. Widmayer: Algorithms and Data structures, Spektrum Akademischer Verlag, 2002
- S. Krumke: Algorithmen und Datenstrukturen, Skript, TU Berlin, WS 2002/03
- K. Mehlhorn and P. Sanders: Algorithms and Data Structures: The Basix Toolbox, Springer, 2008

Overview

Introduction Fibonacci

Introduction Fibonacci-Heaps

Realisation

Amortized Analysis

Experimental Comparison

Leonardo von Pisa

- Fibonacci = „Son of Bonacci“
- ca. 1170 – 1240, Pisa
- Mathematician
- Liber Abaci 1202

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

$F(1) = 1$

$F(2) = 1$

- $\Phi=(1+\sqrt{5})/2 = 1{,}6180339887\ldots$
- $\underline{\Phi}=(1-\sqrt{5})/2 = -0{,}6180339887\ldots$ (Konjugierte)
- $\Phi^{-1}=2/(1+\sqrt{5})=-\underline{\Phi}=(\sqrt{5}-1)/2=0{,}6180339887\ldots$
- Φ^{-1}: „golden ratio“

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

Priority Queues

- How can we realize Priority Queues?
  - Applications, e.g.,
    - shortest paths in Dijkstra
    - minimum spanning trees with Prim
    - finding minimum cuts

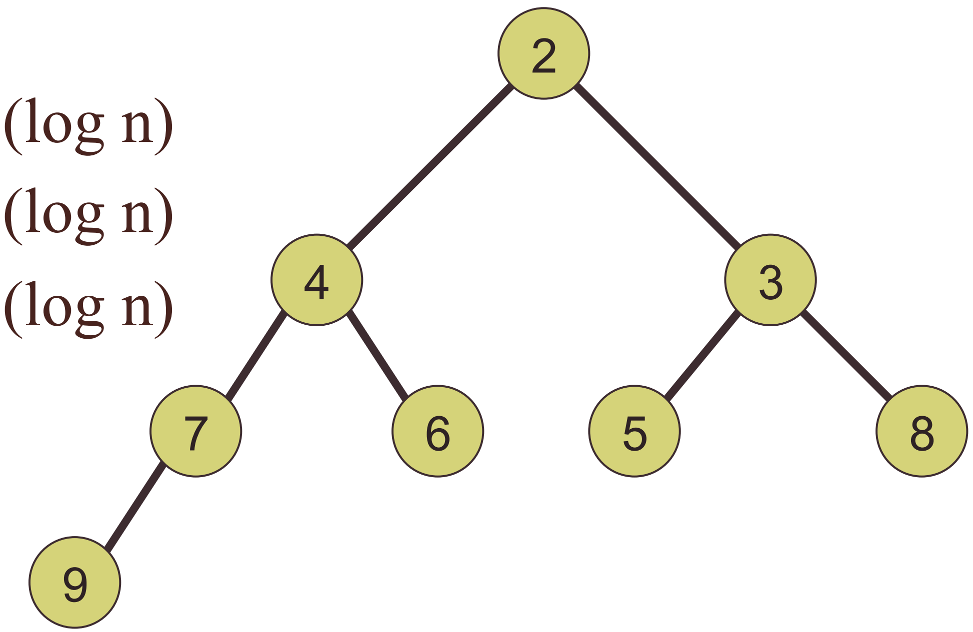
Operationen:

- **Insert**
- **ExtractMin**
- **DecreaseKey**

Binary Heaps

Heap property: priority (key) of parent is not larger than those of children

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



n Knoten

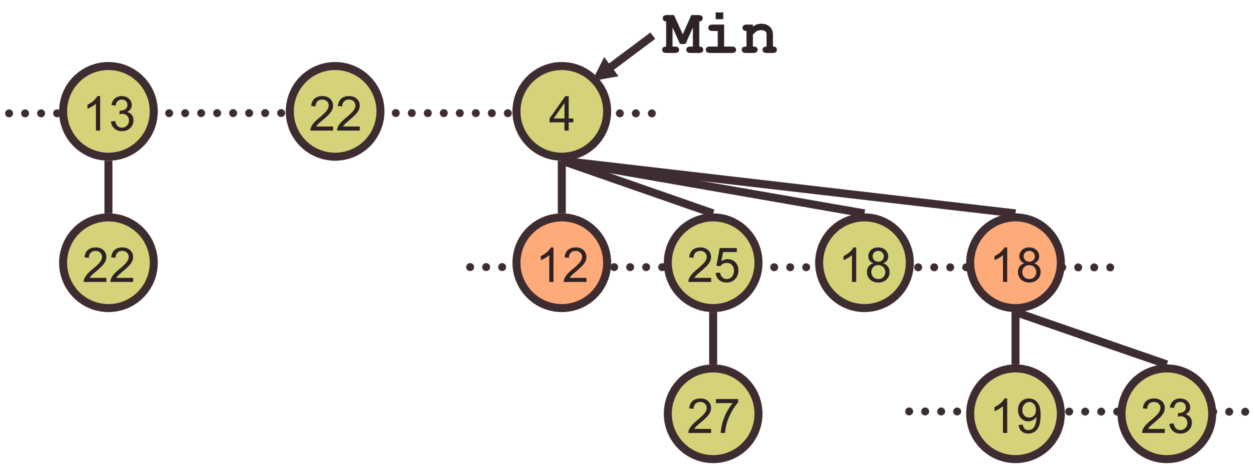
Fibonacci Heaps (Fredman & Tarjan 1987)

Operation	Bin. Heap	Fibonacci-Heap
`Insert`	O(log n)	O(1)
`ExtractMin`	O(log n)	O(log n)*
`DecreaseKey`	O(log n)	O(1)*

* Not really fair, since amortized analysis

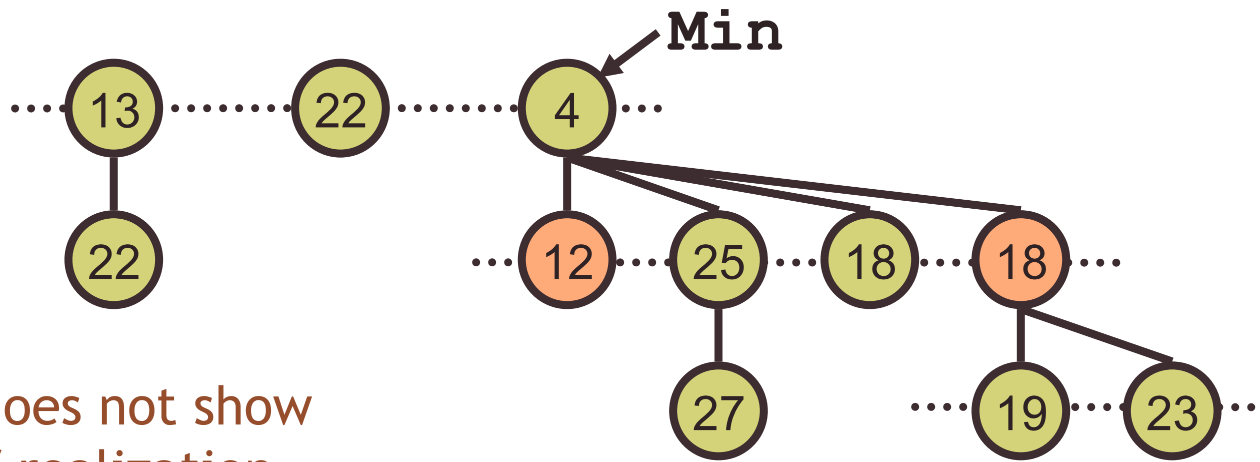
Fibonacci-Heaps: Features

- Set of trees with heap property
- Global `Min`-pointer
- Trees are not necessarily binary
- Each vertex is either marked or unmarked



Fibonacci Heaps: Realization

- All roots are connected via doubly-linked cyclic list: root list
- Each vertex has a pointer to ist parent and to (only) one of ist children (**`parent, child`**)
- All children are connected in a doubly-linked cyclic list.
- Each vertex has an entry for **`key, mark, rank`** (number of children)



Picture does not show links of realization

`Merge(h_1,h_2)`

- Merge two F-heaps h_1 and h_2:
  - Combine both root lists into one
  - The **`Min`** pointer then points to the min of both **`Min`** pointers.

- Time: O(1).

Insert(v)

- Insert(v) into F-Heap h:
  - Build a F-Heap h´ consisting of node v (**mark**=0, **rank**=0)
  - Merge h and h´ a new F-Heap

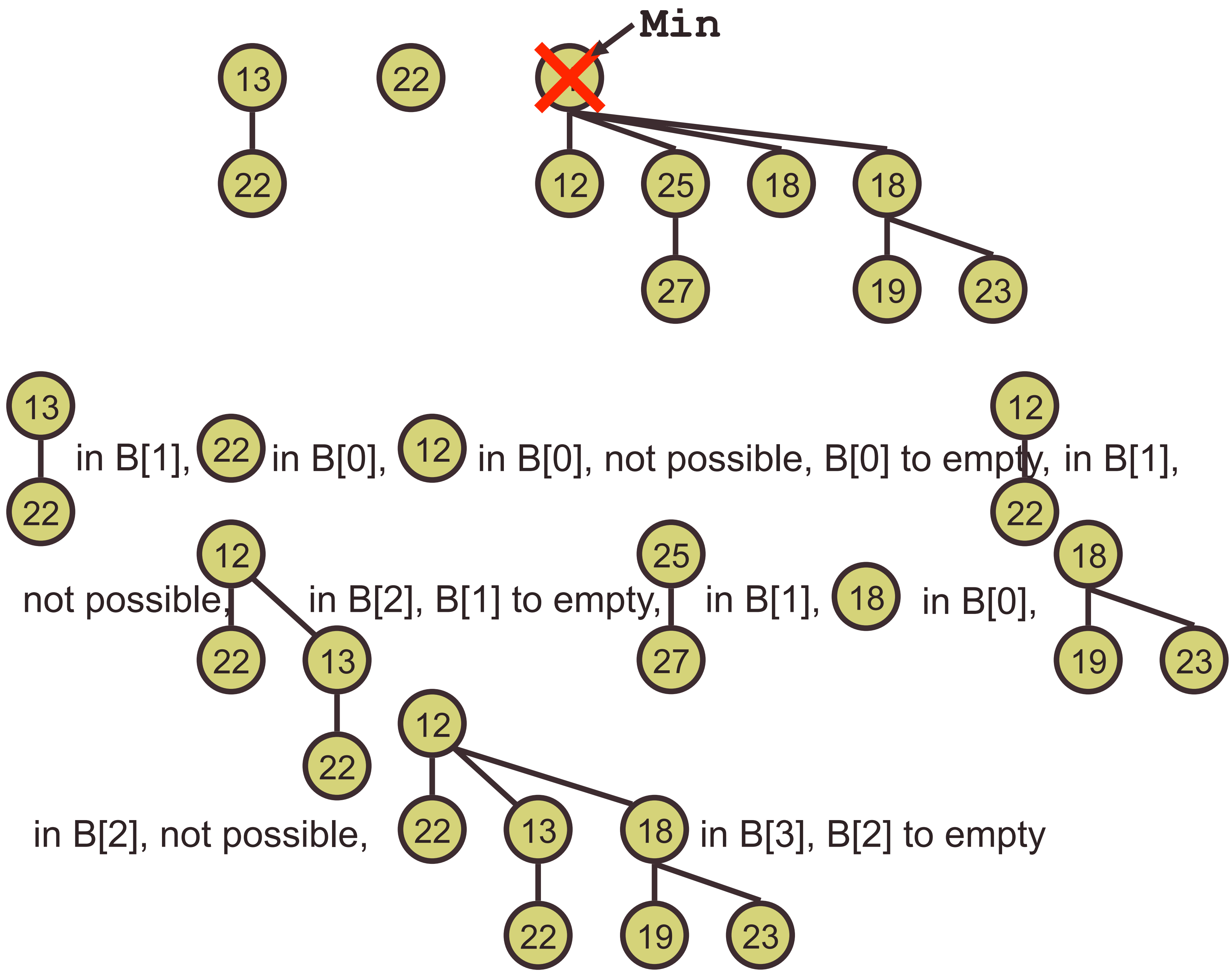
- Time: O(1). But laziness takes its revenge on **ExtractMin**...

Fibonacci-Heaps: `ExtractMin`

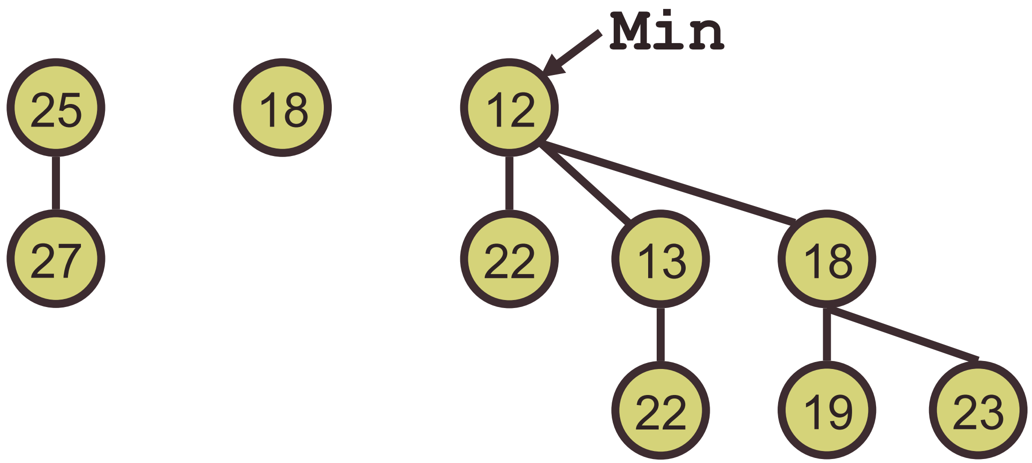
- It is trivial to find the Minimum (**`Min`**-Pointer)
- Delete the associated node from the root list
- Append all the children from the Min to the root list
- We need to traverse all nodes in the root list in order to find the new Minimum
- **Idea:** If we have to do that, we can also clean up.
- We make sure that no two roots have the same degree (**`rank`**)
  - For this we use an array B[0..n-1] with the interpretation B[i] = v $\leftrightarrow$ **`rank`**(v) = i

Clean Up Procedure for `ExtractMin`

- For all roots v:
  - Set **`mark`**(v)=0 // all roots are unmarked
  - d = **`rank`**(v)
  - **If** B[d] empty, then set B[d] = v
  - **Else** {
  - **Repeat**
    - make larger root the child of the smaller
    - set B[d] = empty // empty again
    - d = d+1
  - **Until** B[d] empty // found an empty position for d
  - B[d] = smaller root// **`rank`**(v) is now d
  - }


Min
in B[1],
in B[0],
in B[0], not possible, B[0] to empty, in B[1],
not possible,
in B[2], B[1] to empty,
in B[1],
in B[0],
in B[2], not possible,
in B[3], B[2] to empty

Running Time for **`ExtractMin`**



- Time: $O(d + W_1 + m) = O(d + W_1)$,
  - where $d = \mathrm{deg}$**(**deleted root**)**
  - and W_1 = #roots before deletion
  - and m = number of *merge* operations (clean up)
- since: at least 1 merge-call for each root
- detailled amortized analysis: later

Fibonacci-Heaps: `DecreaseKey`

- **DecreaseKey(**v, newKey**):**
  - decrease the key of v to value newKey;
  - **If** Heap-property is violated, then **{**
  - **Repeat:** //separate par(v) ab
    - Separate v from p=**`parent`**(v);
    - Insert v into root list; set **`Mark`**(v)=0;
    - Decrease **`rank`(**p)
    - Set v=p;
  - **Until** **`mark`**(p)==0 or **`p`** gets into root list
  - **If** p is not in root list, then set **`mark`**(p)=1;

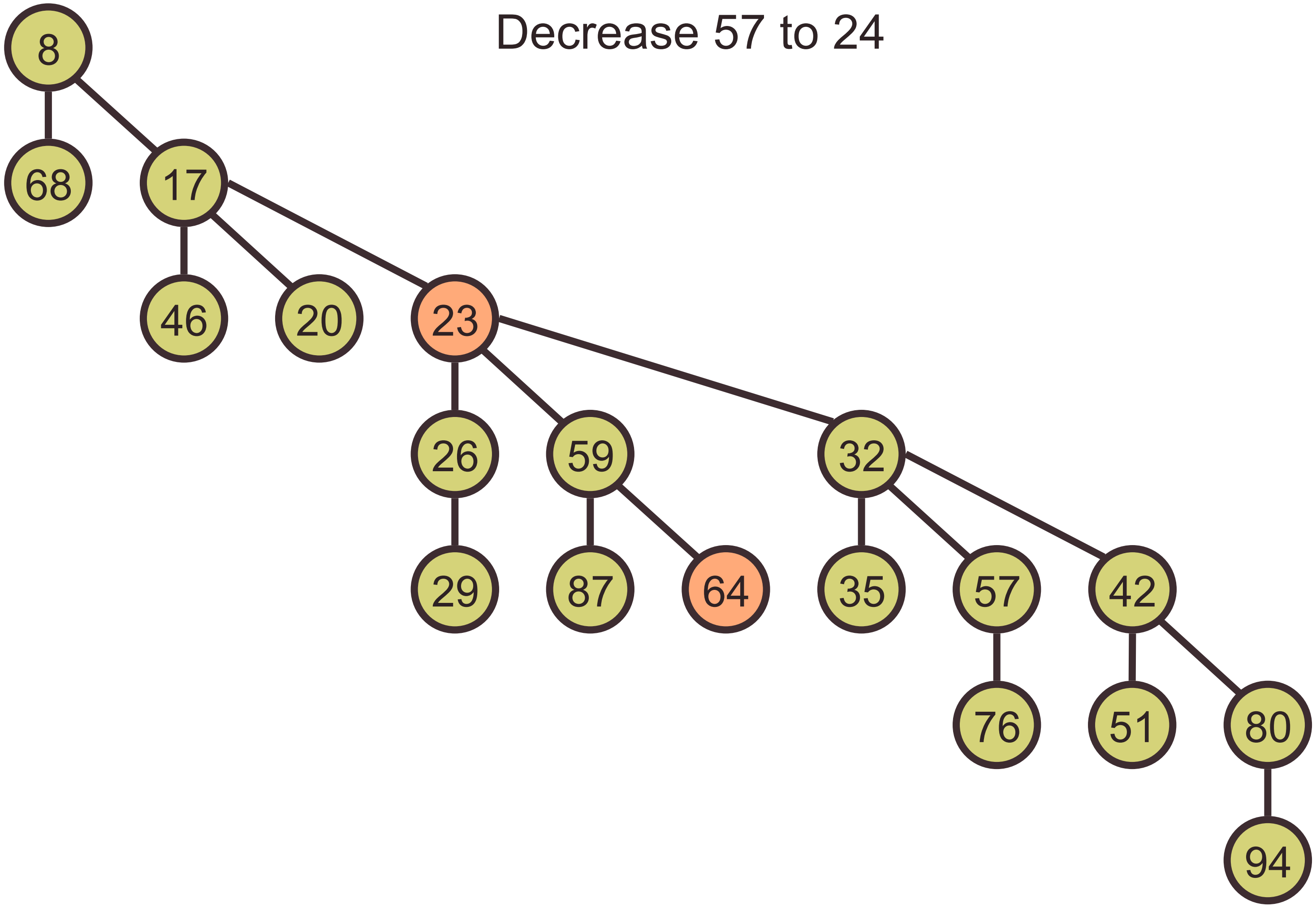
 }

  - Update **`Min`**-pointer

Marking prevents F-Heaps from becoming too thin

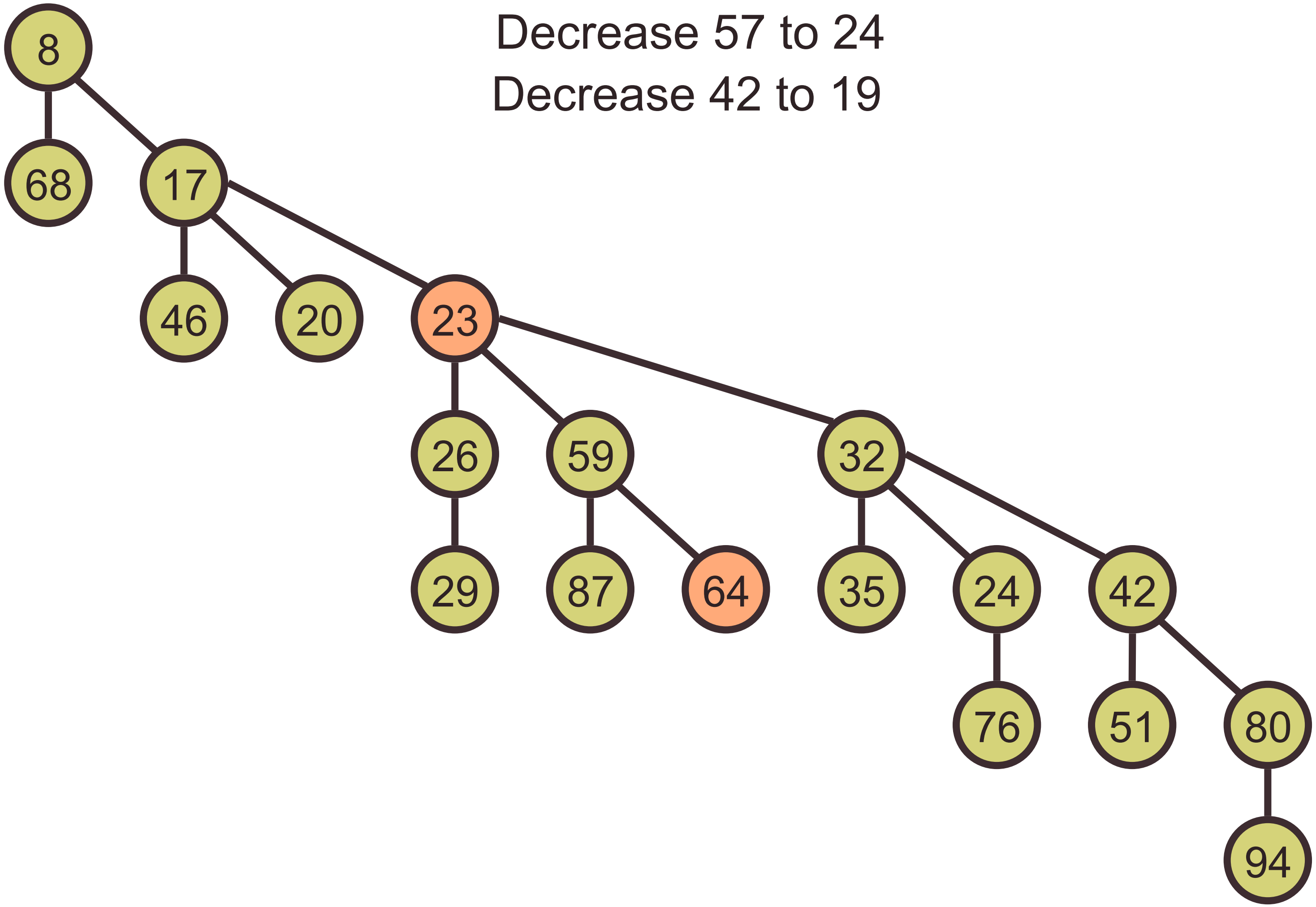
Fibonacci-Heaps: `DecreaseKey`

Decrease 57 to 24



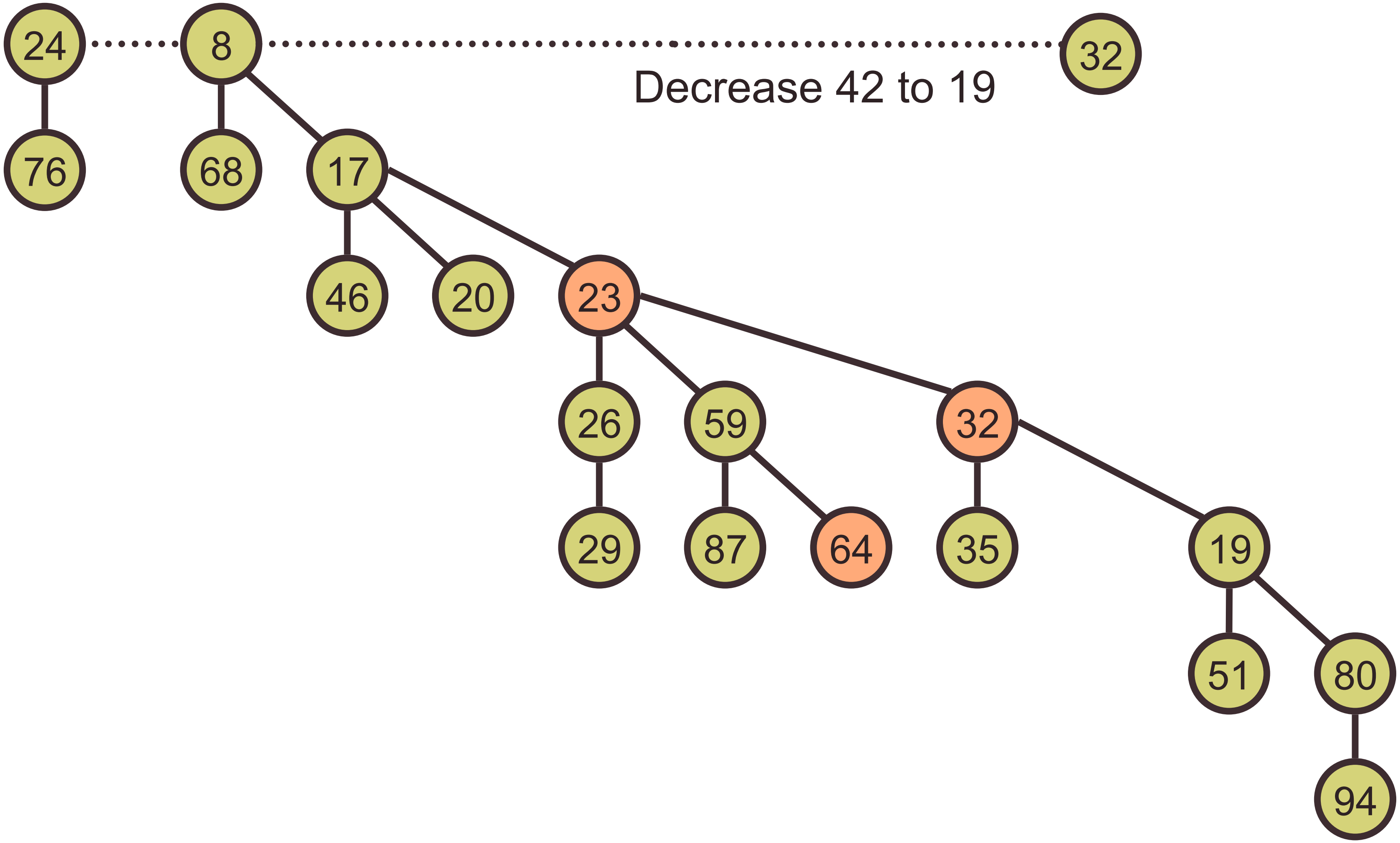
Fibonacci-Heaps: `DecreaseKey`

Decrease 57 to 24
Decrease 42 to 19



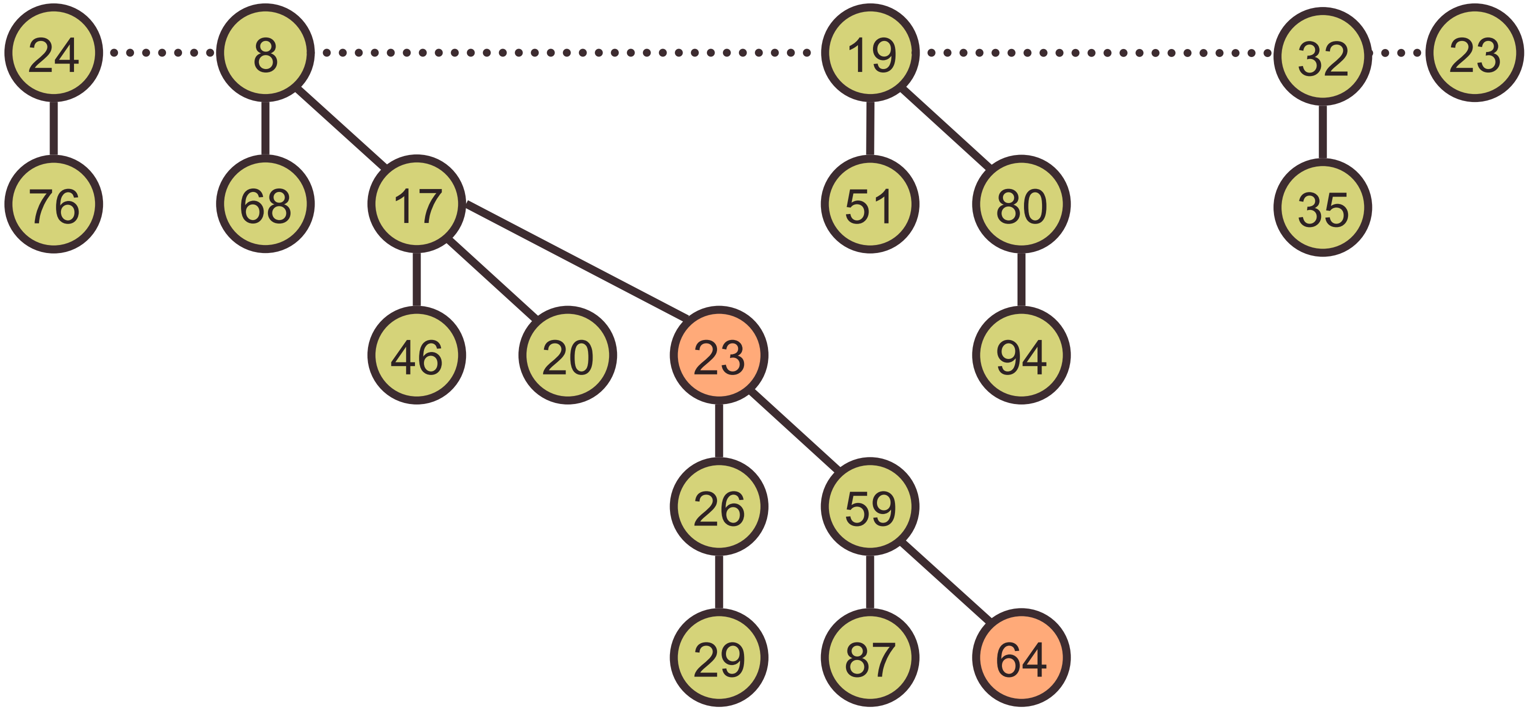
Animation

Fibonacci-Heaps: `DecreaseKey`



Animation

Fibonacci-Heaps: `DecreaseKey`



Decrease 42 auf 19

Animation

Fibonacci-Heaps: Analysis

We want to show that the node degrees in the F-Heap are bounded by $O(\log n)$

Lemma F1: Let p be a node of degree d and let $v_1,\dots,v_d$ be the children of p in their order in which they have been appended to p (by merge). Then we have:

The i-th child of p has degree at least i-2.

Proof: When the i-th child v_i was appended to p, the nodes v_i and p have had the same degree which was at least i-1.

Since then, v_i could have lost at most one child; so its degree is still at least i-2.

Fibonacci-Heaps: Analysis

Lemma F2: Each node p of degree k of a F-Heaps is root of a subtree with at least F_{k+2} nodes.

Remark: F_{k+2} is the (k+2)-th Fibonacci number. Because of this, Fredman and Tarjan have given the name Fibonacci-Heap.

Proof of Lemma F2

Let p be a node of degree k and let $v_1,\ldots,v_k$ the children of p in the order in which they have been appended to p.

Let S_k be the minimum number of all the successors of node p (including p).

We show: $S_k \geq F_{k+2}$

Because of Lemma F1 we know:

v_1 had at least degree 0: contribution to S_k: $S_0 \geq 1$ ✔

v_2 had at least degree 1, and now at least 0: $S_0 \geq 1$ ✔

v_3 had at least degree 2, and now at least 1: $S_1 \geq 2$ ✔

v_k had at least degree k-1, and now at least degree k-2: S_{k-2} ✔

$$\Rightarrow S_k \geq \underbrace{1}_{\text{from } p} + \underbrace{S_0}_{\text{from } v_1} + S_0 + S_1 + \ldots + S_{k-2} \geq 1 + 1 + \sum_{i=0}^{k-2} S_i \quad \text{for } k \geq 2$$

Proof of Lemma F2 ff

We have shown: $$S_k \geq 1 + 1 + \sum_{i=0}^{k-2} S_i$$

Claim: From Def. of Fibonacci numbers: $$F_{k+2} = 1 + 1 + \sum_{i=2}^{k} F_i$$

Proof of Claim: We show for k≥0: $$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

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

Induction Hypothesis: for all k up to k+1 already shown

We show it for k+2:

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

Proof of Lemma F2 fff

We have shown:
$$S_k \geq 1 + 1 + \sum_{i=0}^{k-2} S_i$$
$$F_{k+2} = 1 + 1 + \sum_{i=2}^{k} F_i$$

Still to show: $S_k \geq F_{k+2}$

- k=0: $S_0 = 1 \geq F_2 = 1$ ✔
- k=1: $S_1 \geq 2 \geq F_3 = 2$ ✔
- k≥2: Induction Hypothesis: $S_i \geq F_{i+2}$ is valid for $i \leq k-1$:

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

Fibonacci-Heaps: Analysis

We have shown

Lemma F2: Each node p of degree k of a F-Heaps is root of a subtree with at least F_{k+2} nodes.

Lemma F3: Each node of a F-Heaps with N nodes has a degree of at most $k \leq \log_{\Phi} N$.

Proof: We have $F_{k+2} \geq \Phi^k$ (Goldene Schnitt) (without proof)

$$N \geq S_k \geq F_{k+2} \geq \Phi^k \quad \Rightarrow \quad k \leq \log_{\Phi} N$$

Corollary: After `ExtractMin` the root list contains at most O(log N) roots.

Fibonacci-Heaps: Analysis

- Oops: n times `insert`, 1 `ExtractMin` → $\Theta(n)$
  - Worst case: even worse than binary heaps
  - But we will see: rarely so bad
  - Intuition: `Insert` builds up tension/potential, which makes `ExtractMin` slow, but afterwards the potential decreases.Future `ExtractMin` will have it easier...
  - Potential is increasing slowly (over n Insert-operations) → average costs over all operations are **O(1)**.
- We need amortized analysis
  - Watch out: Not useful for real time problems!

Fibonacci-Heaps: Amortized Analysis

- If any sequence of operations is executed, the total time required for this is limited by the total amortized time.

- We show: The amortized time of a single **ExtractMin** operation is $O(\log n)$, the amortized time of all **other** operations is $O(1)$.

Amortized Analysis via Potential Method

- We describe the potential of the data structure mathematical:
- Function Φ, which maps each heap to a real number
- **Amortized costs** of an operation are defined as **real costs + $\Delta\Phi$**
- Thus also burdened by future costs (`Insert`)
  or defusing costly operations ($\Delta\Phi$ negative).
- Let $o_1,\ldots,o_n$ a sequence of n operations
  - a_i amortized costs for operation o_i
  - t_i real costs für Operation o_i
  - Φ_i potential directly after operation o_i
  - Φ_0 potential directly before o_1

Amortized Analysis via Potential Method

- Then the sum of the amortized costs is:

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

- We choose a potential function, which is
  - Non-negativ and
  - In the beginning equal to 0.
- Then: $\Phi_n - \Phi_0 \geq 0$ and hence $\boldsymbol{\Sigma t_i \leq \Sigma a_i}$
- (the total amortized costs provide a bound for the total real costs)

Amortized Analysis: How to choose Φ

- What can cause big costs later?
- Certainly the number of roots W
- We will make a provisional choice first.: $\Phi = \alpha W$
- *(will be extended soon...)*

Amortized Analysis: `ExtractMin`

- Deleted node had degree **d**, before the operation, the F-Heap had $\mathbf{W_1}$ roots, after **`ExtractMin`** it had $\mathbf{W_2}$ roots, for cleanup **m** *merge* operations needed
- **We have:** $\mathbf{W_2 = W_1 + d - m}$ and $\mathbf{m \leq W_1 + d}$ *(because with every merge a root disappears)*
- Real costs $\mathbf{t_i = c(W_1+d)}$, **c** const.
- $\mathbf{\Delta\Phi = \alpha(W_2 - W_1)}$, thus amortized costs:
  $\mathbf{a_i = c(W_1+d) + \alpha(W_2 - W_1) = (c - \alpha)W_1 + cd + \alpha W_2}$
- We choose $\mathbf{\alpha = c}$, thus $\mathbf{a_i = c(d + W_2)}$
  - but $\mathbf{d = O(\log n)}$ and $\mathbf{W_2 = O(\log n)}$, because of the degrees of all nodes are bounded by $\mathbf{O(\log n)}$
  - also $\mathbf{a_i = O(\log n)}$

Amortized Analysis: `DecreaseKey`

- creates **k** new roots
- Real costs $t_i = c'(k + 1)$, c' constant
- **Problem:** This is expensive and increases the potential...
- **Solution:** Second source for Φ: **A***:* number of marked nodes
  - Become easy to roots and still cause work, therefore **new Potential function:** $\Phi = \alpha W + \beta A$
- $\Delta\Phi \leq \alpha k + \beta(-(k-1)+1)=\alpha k + \beta(2 - k)$, because of all new roots were marked (maybe up to the last), at most one node get newly marked
- Amortized costs of `DecreaseKey`:
- $a_i \leq c'(k + 1) + \alpha k + \beta(2 - k) = (c' + \alpha - \beta)k + c' + 2\beta$
- Choose $\beta = c' + \alpha$, then we get $a_i \leq 3c' + 2\alpha = O(1)$

Amortized Analysis: `ExtractMin` (new)

- Deleted node had degree $\mathbf{d}$, before the operation, the F-Heap had $\mathbf{W_1}$ roots, after `ExtractMin` it had $\mathbf{W_2}$ roots, for cleanup $\mathbf{m}$ *merge* operations needed
- **We have:** $W_2 = W_1 + d - m$ and $m \le W_1 + d$ *(because with every merge a root disappears)*
- Real costs $t_i = c(W_1+d)$, $\mathbf{c}$ const.
- $\Delta\Phi = \alpha(W_2 - W_1) + \beta\Delta A$, thus amortized costs:
  $a_i = c(W_1+d) + \alpha(W_2 - W_1) + \beta\Delta A \le (c - \alpha)W_1 + cd + \alpha W_2$
- *(because nodes may only lose their mark)*
- We choose $\alpha = c$, thus $a_i = c(d + W_2)$
  – but $d = O(\log n)$ and $W_2 = O(\log n)$, because of the degrees of all nodes are bounded by $O(\log n)$
  – also $a_i = O(\log n)$

Amortized Analysis: `Insert`

- **Insert**:
  - Increases the number of roots to 1
  - $\boldsymbol{a_i = t_i + \alpha = O(1+\alpha) = O(1)}$
- Obviously, **Φ is non-negative** and in the beginning **0.**

Note: The amortized costs of all operations must be estimated using the same potential function.
(if you like to analyze a sequence of operations in amortized time.)

Beweis-Ende

Overview

Introduction Fibonacci

Introduction Fibonacci-Heaps

Realization

Amortized Analysis

Experimental Comparison

Experimental Comparison

- f_heap: Fibonacci-Heaps
- p_heap: pairing heaps — similar to F-Heap, tree, rebalancing by pairwise merge at **ExtractMin, s. Übung**
- k_heap: k-ary heaps
- bin_heap: binary heaps
- list_pq: linear unsorted list
- r_heap: radix heaps (AMOT90)* (also called redistributive)
- m_heap: monotone Heaps*

*: only monotone heaps (e.g., for Dijkstra) and restricted to bounded integers as key/priority

Theoretical Running Times of the PQs for Dijkstra

- Graph G with n vertices, m edges and costs <=M
- Worst Case: n **`Insert`**, m **`DecreaseKey`** operations
- Expected for randomly generated G with random weights:
  - only n **`Insert`** and O(min(m, n log(2m/n)) **`DecreaseKey`** Ops

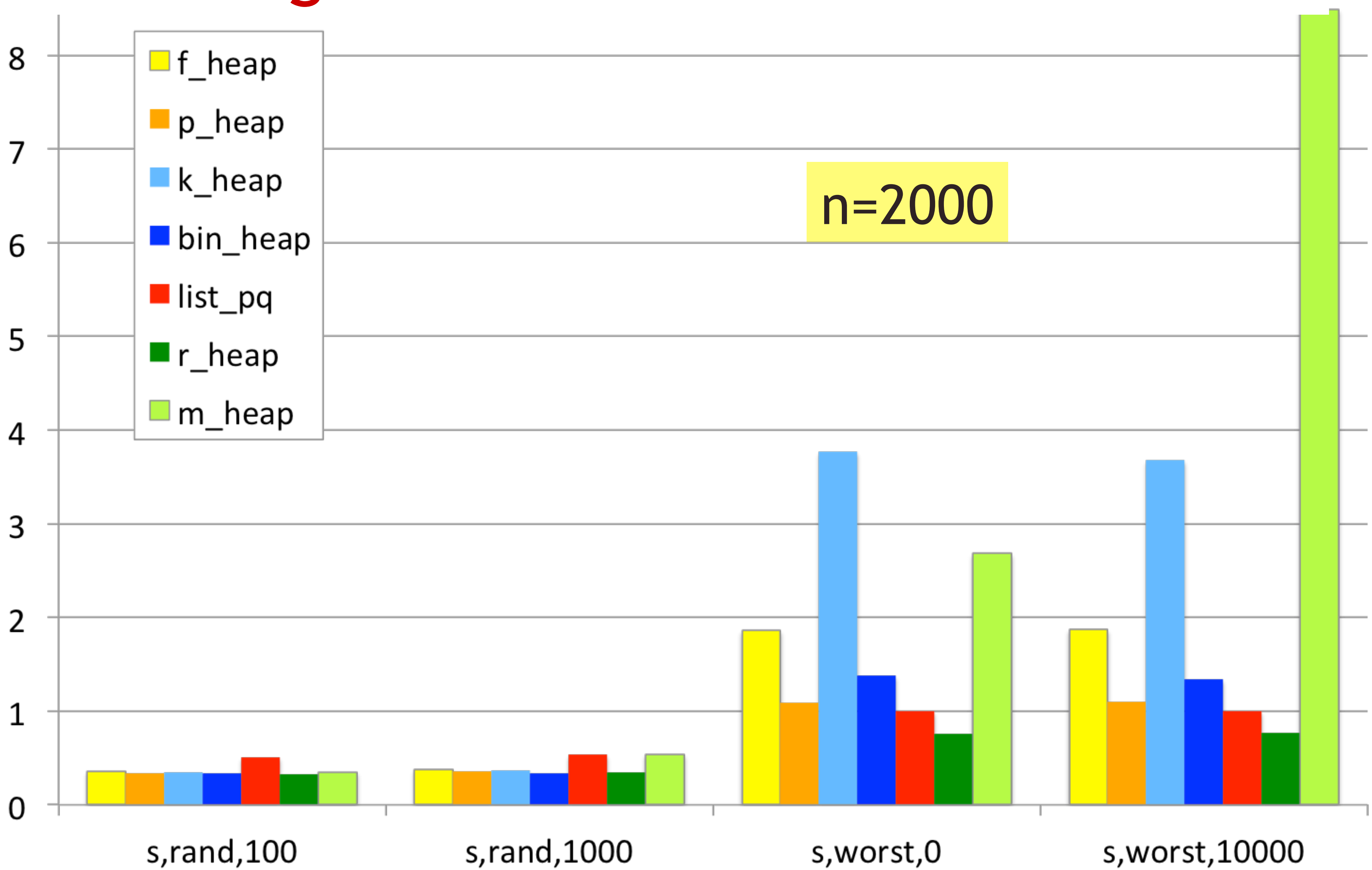
PQ	Worst Case Laufzeit	Erwartete Laufzeit
F_heap	$O(m + n \log n)$	$O(m + n \log n)$
p_heap	$O(m + n \log n)$	$O(m + n \log n)$
k-heap	$O(m \log_k n + nk \log_k n)$	$O(m + n (\log(m/n) + k) \log_k n)$
bin_heap	$O(m \log n + n \log n)$	$O(m + n \log(m/n) \log n)$
list_pq	$O(m + n^2)$	$O(m + n^2)$
r_heap*	$O(m + n \log M)$	$O(m + n \log M)$
m_heap*	$O(m + maxdist + M)$	$O(m + maxdist + M)$

*: costs restricted to bounded integers

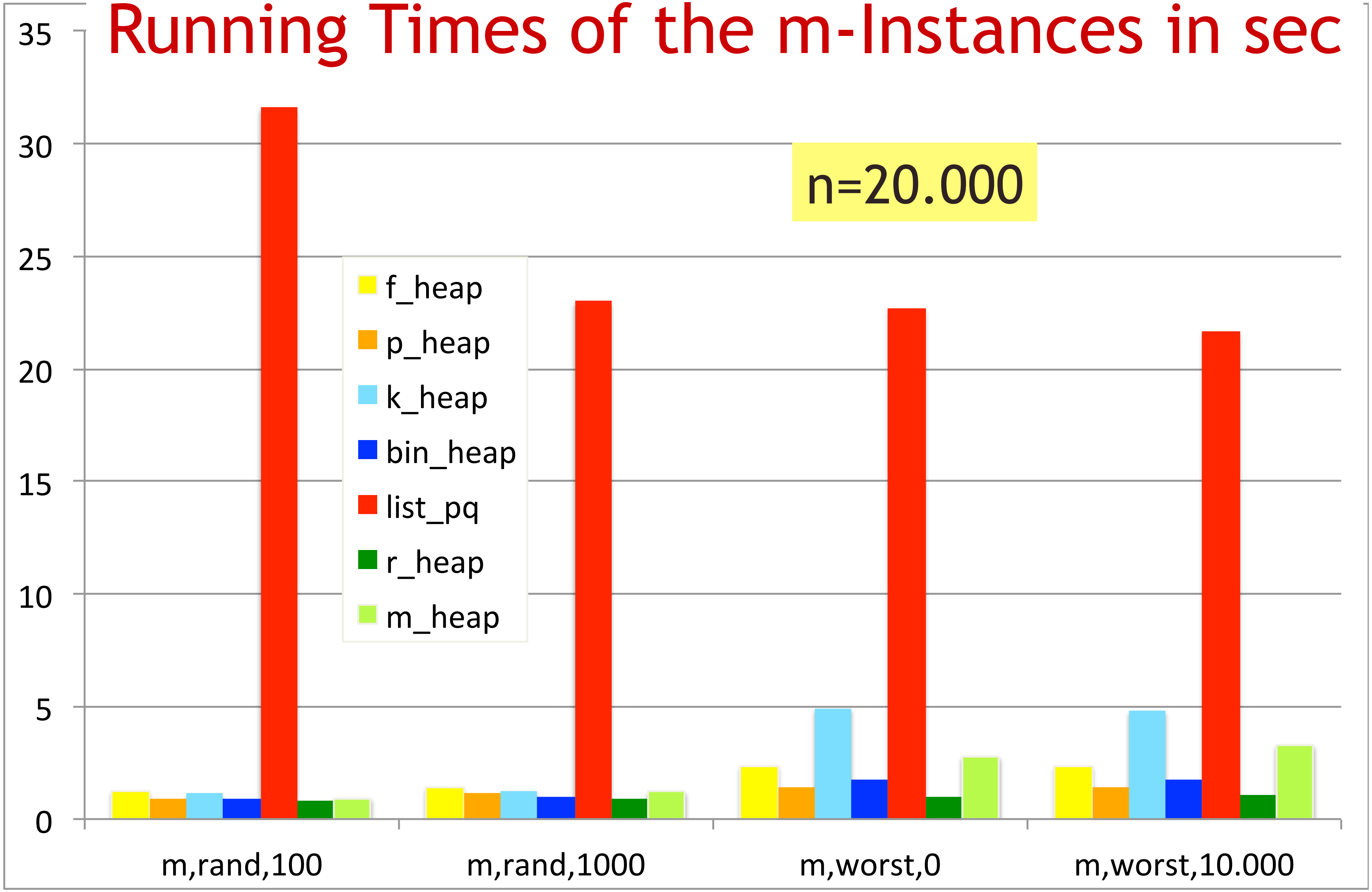
Experimental Comparison

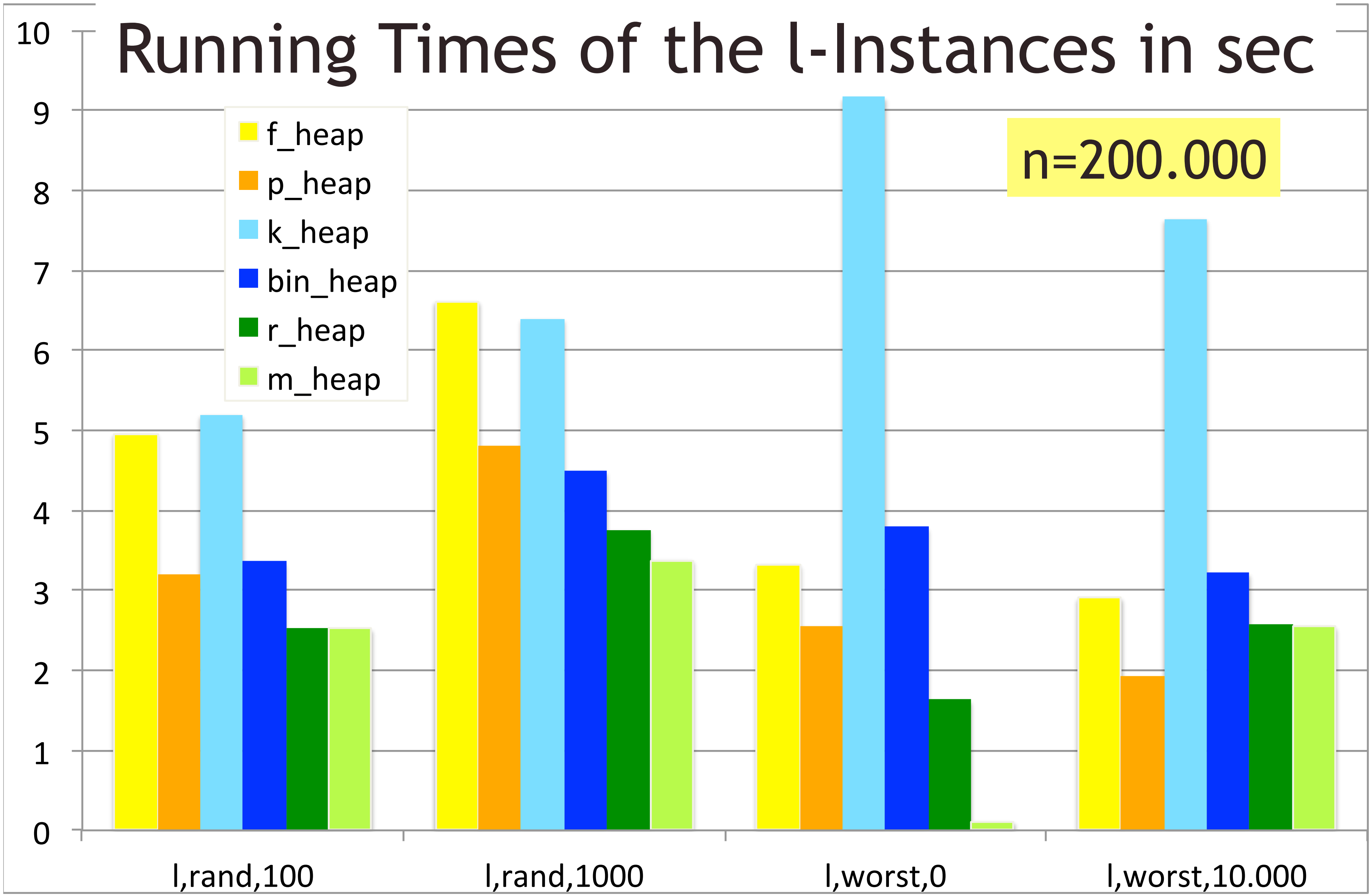
- Mehlhorn & Näher: LEDA book 2000
- comparison within Dijkstra
- vertex number n: 2000 (s), 20.000 (m) und 200.000 (l)
- m=500.000 edges
- 4 instances each for s, m und l:
- Edge weights: random (rand) from [0..M-1] for M=100 and M=1000
- worst-case graph (worst) and edge weights (w), so that the update-time is maximum (costs for subgraph c=0 and c=10.000, remaining edges are chosen so that the number of `DecreaseKey` operations is m-n+1)
- k-ary heaps with k=max(2,m/n)

Running Times of the s-Instances in sec



Running Times of the m-Instances in sec




Running Times of the l-Instances in sec
n=200.000
f_heap
p_heap
k_heap
bin_heap
r_heap
m_heap
0
1
2
3
4
5
6
7
8
9
10
l,rand,100
l,rand,1000
l,worst,0
l,worst,10.000

Conclusions from the Experimental Comparison

- r_heaps and m_heaps are the fastest.
- They only work for integer costs and they are only allowed for monotone PQs, i.e., the key of each **`Insert`** or **`DecreaseKey`** operation is at least as large as the last **`ExtractMin`** Operation. This is true for Dijkstra but not for PRIM (for Minimum Spanning Tree).
- The safest and fastest in practice are p_heaps or bin_heaps.
- If the number of **`DecreaseKey`** operations is large, then **p-heaps** are preferable.

END F-Heaps