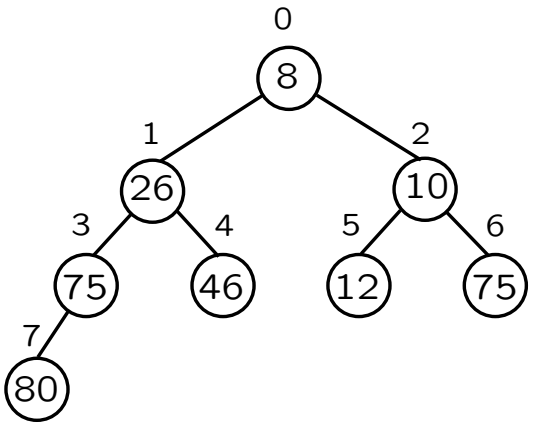
# Towards Ultimate Binary Heaps

**Jyrki Katajainen**[1,2]

**Amr Elmasry**[3]

[1] University of Copenhagen
[2] Jyrki Katajainen and Company
[3] Alexandria University

# Binary heaps




$n = 8$

| $a$ | 8 | 26 | 10 | 75 | 46 | 12 | 75 | 80 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

*left-child*$(i)$
    **return** $2i + 1$

*right-child*$(i)$
    **return** $2i + 2$

*parent*$(i)$
    **return** $\lfloor (i - 1)/2 \rfloor$

*construct*()
    **for** $(i = parent(n - 1);\ i \geq 0;\ --i)$
        *sift-down*$(i)$

*minimum*()
    **return** $a_0$

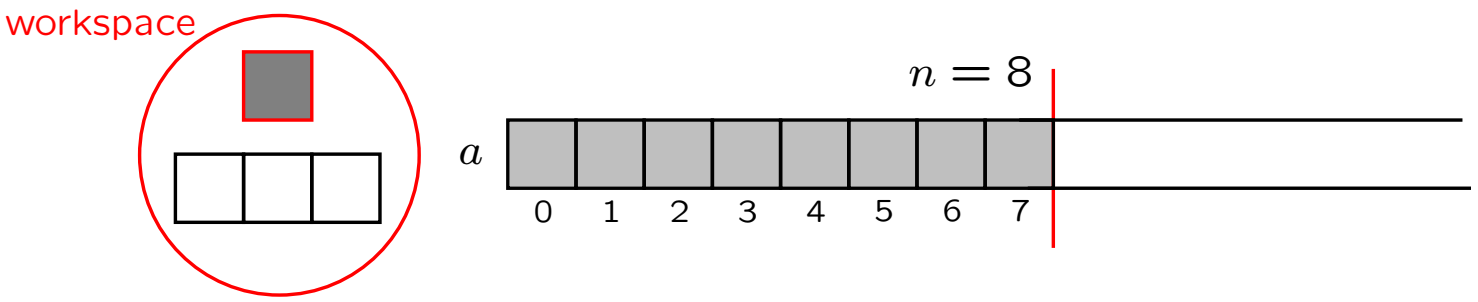
*insert*$(x)$
    $a_n = x$
    *sift-up*$(n)$
    $n$ += 1

*extract-min*()
    $min = a_0$
    $n$ −= 1
    $a_0 = a_n$
    *sift-down*(0)
    **return** $min$

# Model of computation

## Available

- An **infinite** array $a$ suitable for storing elements
- $O(1)$ other memory locations for storing elements
- $O(1)$ variables (counters, indices, bit strings of length $\lceil \lg(1+n) \rceil$)




## Requirement

- If the data structure stores $n$ elements, these elements must be kept in the first $n$ locations of $a$.

# 2014: The 50th anniversary of binary heaps

## [Williams 1964]

| Operation | Worst-case time | # element comparisons |
|---|---|---|
| *minimum* | $O(1)$ | 0 |
| *insert* | $O(\lg n)$ | $\lg n + O(1)$ |
| *extract-min* | $O(\lg n)$ | $2\lg n + O(1)$ |

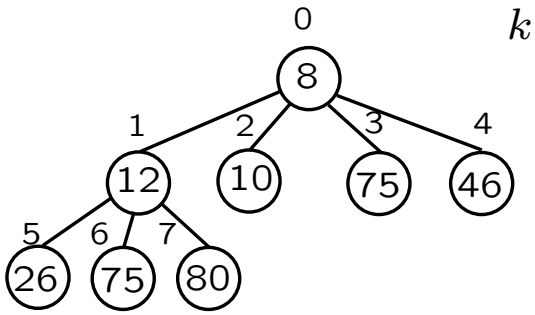
## [Gonnet & Munro 1986]

| Operation | # element comparisons |
|---|---|
| *insert* | $\lg\lg n \pm O(1)$ sufficient and necessary |
| *extract-min* | $\lg n + \log^* n \pm O(1)$ sufficient and necessary |

## Assumptions

1. Single heap
2. Perfectly heap ordered before and after each operation
3. Operations are memoryless.

# Multi-ary heaps




$k = 4$

$n = 8$

| $a$ | 8 | 12 | 10 | 75 | 46 | 26 | 75 | 80 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

*first-child*$(i)$
    **return** $k \times i + 1$

*parent*$(i)$
    **return** $\lfloor (i-1)/k \rfloor$

**[Johnson 1975]**

| Operation | Worst-case time |
|---|---|
| *minimum* | $O(1)$ |
| *insert* | $O(\log_k n)$ |
| *extract-min* | $O(k \log_k n)$ |

$n$: # elements

$k$: arity

# Open—but solved later (see the last slide)

| Operation | Worst-case time | # element comparisons |
|---|---|---|
| *minimum* | $O(1)$ | 0 |
| *insert* | $O(1)$ | $O(1)$ |
| *extract-min* | $O(\lg n)$ | $\lg n + O(1)$ |

in-place as binary heaps

## Sorting

| Operation | Worst-case time | # element comparisons |
|---|---|---|
| *minimum* | $O(1)$ | 0 |
| *insert* | $O(1)$ | $O(1)$ |
| *extract-min* | $\Omega(\lg n)$ | $\lg n - O(1)$ necessary |

$n$: # elements

# Related work

**[Carlsson, Munro & Poblete 1988]**

queue of pennants
$extract\text{-}min$: $3 \lg n + \log^* n$ element comparisons

**[Elmasry, Jensen & Katajainen 2008]**

multipartite priority queue
$O(n)$ extra words

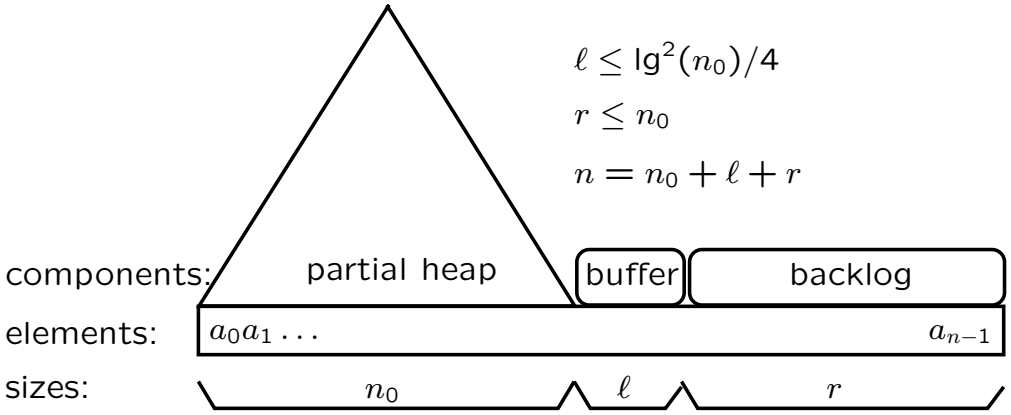
**[Edelkamp, Elmasry & Katajainen 2012]**

engineered weak heaps
$n + O(w)$ extra bits

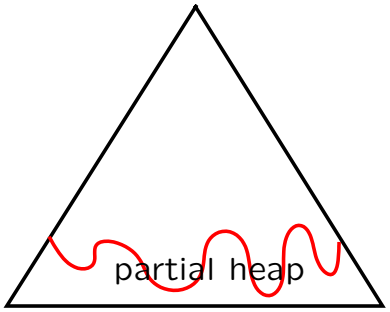
$n$: # elements
$w$: word size in bits

# Our main result

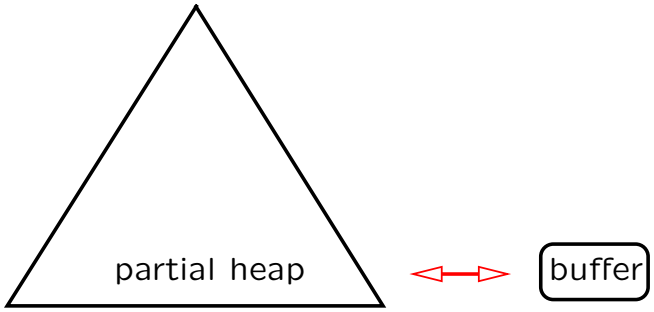
The objective can be achieved in the amortized sense




## What is special?




heap-order violations

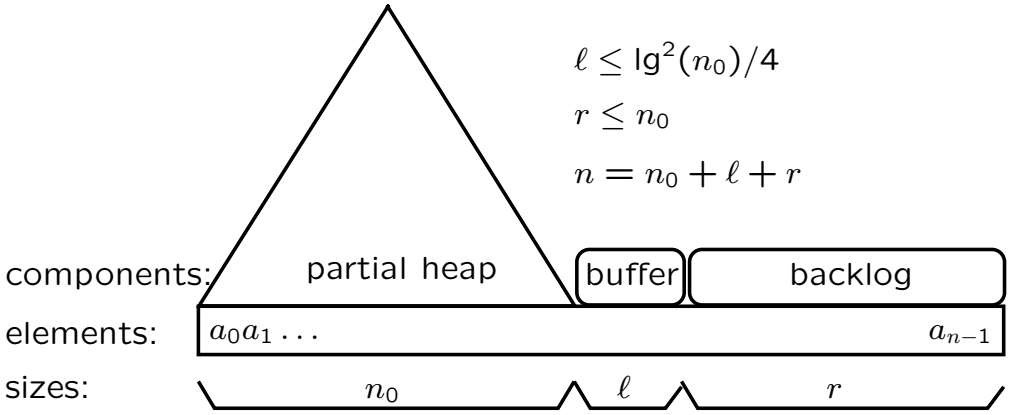



heap-order violations

backlog

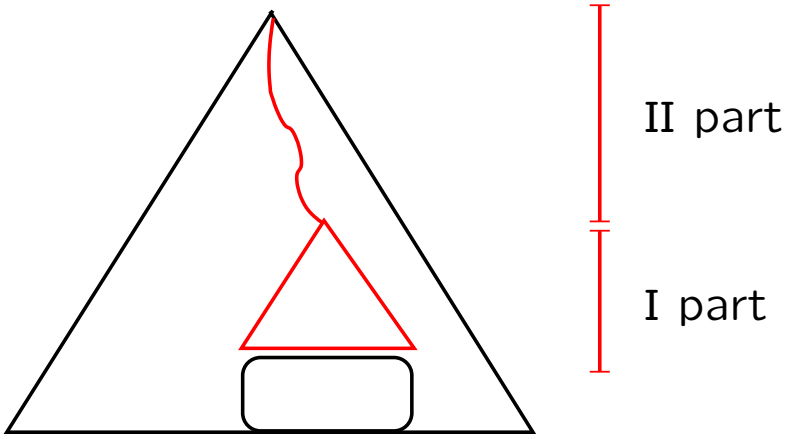
some large elements kept outside

# Insert




$insert(x)$
  $a_n \leftarrow a_{n_0+\ell}$
  $a_{n_0+\ell} \leftarrow x$
  **if** $(\ell = 0)$
    $k \leftarrow \lceil \lg(n_0)/2 \rceil$
  $k\text{-}ary\text{-}heap\text{-}insert(a, n_0, \ell, a_{n_0+\ell})$
  $\ell$**++**
  $n$**++**
  **if** $(\ell = k^2)$
    $bulk\text{-}insert(a, n, n_0, \ell)$

# Bulk-insert



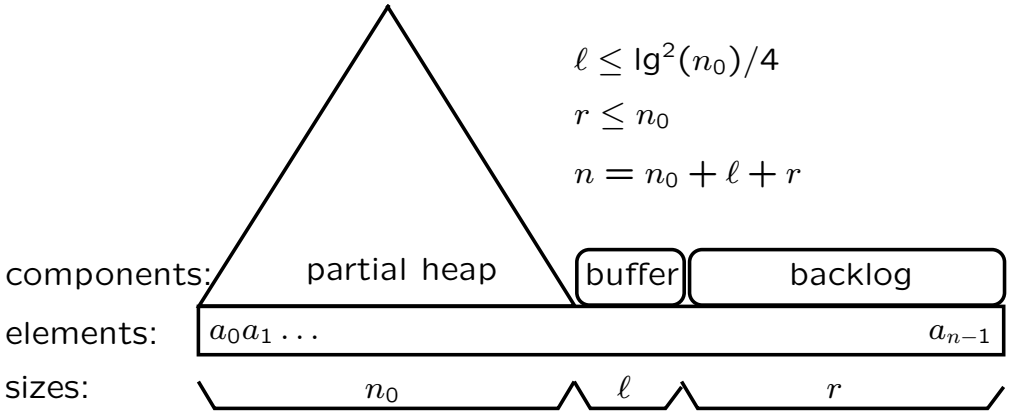

Use Floyd's heap-construction algorithm in the red area: call $sift\text{-}down$ at each node

I part: $O(\ell)$ nodes; $O(\ell)$ total work as for Floyd's algorithm

II part: $O(\lg(n_0))$ $sift\text{-}down$ calls $\Rightarrow O(\lg^2(n_0))$ total work

$\therefore$ amortized $O(1)$ work per $insert$

# Extract-min




## Minimum in the buffer

$k\text{-}ary\text{-}heap\text{-}extract\text{-}min(a, n_0, \ell)$
$a_{n_0+\ell} \leftarrow a_n$
$\ell$--
$n$--

## Minimum in the heap

$extract\text{-}root(a, n, n_0, \ell)$
$n$--

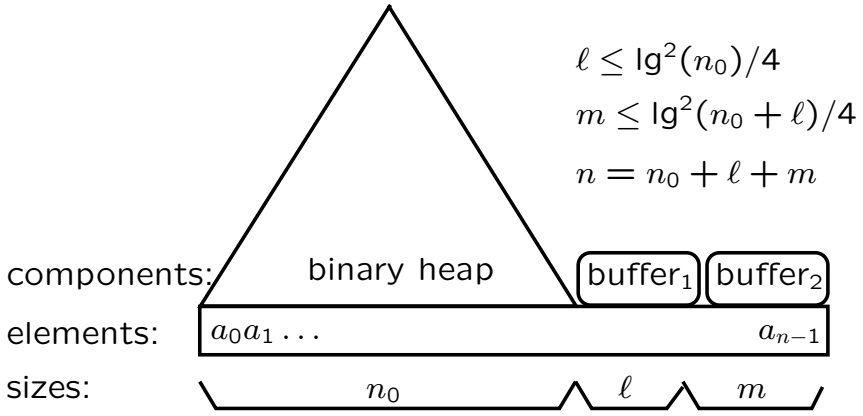
$extract\text{-}root(a, n, n_0, \ell)$

**if** $(n_0 + \ell = n)$
  $rebuild\text{-}structure(a, n, n_0, \ell)$
$i \leftarrow 0$
**while** $(i \leq parent(n_0 - 1))$
  **if** $(a_{left\text{-}child(i)} < a_{right\text{-}child(i)})$
    $a_i \leftarrow a_{left\text{-}child(i)}$
    $i \leftarrow left\text{-}child(i)$
  **else**
    $a_i \leftarrow a_{right\text{-}child(i)}$
    $i \leftarrow right\text{-}child(i)$
$a_i \leftarrow a_{n-1}$

# Deamortization of insertions

The objective can **almost** be achieved in the worst case

| Operation | Worst-case time | # element comparisons |
|---|---|---|
| *minimum* | $O(1)$ | $O(1)$ |
| *insert* | $O(1)$ | $O(1)$ |
| *extract-min* | $O(\lg n)$ | $\lg n + \log^* n + O(1)$ |




## Role of the buffers

- buffer$_2$ takes care of the inserted elements
- buffer$_1$ is incrementally submerged into the heap by performing a constant amount of work per *insert*

# Online sorting

| Operation | Worst-case time | # element comparisons |
|---|---|---|
| *construct* | $O(n)$ | $O(n)$ |
| *extract-min* | $O(\lg n)$ | $\lg n + O(1)$ |

$\rightarrow$ Read the details from the paper

**[Elmasry 2003]**

$$O(n) \text{ extra bits}$$

$n$: # elements

# Note added on 15 April, 2013

Stefan Edelkamp came with the missing link that helped us to solve the problem. We are in the process of writing the paper "Optimal in-place heaps". The new construction shows that the both lower bounds proved by Gonnet & Munro can be bypassed.