CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Priority Queues and Heaps

Priority queue operations

- M.build ($\{e_1, \ldots , e_n\}$)

  M := $\{e_1, \ldots, e_n\}$

- M.insert(e)

  M := M $\cup$ {e}

- M.min

  return min M

- M.deleteMin

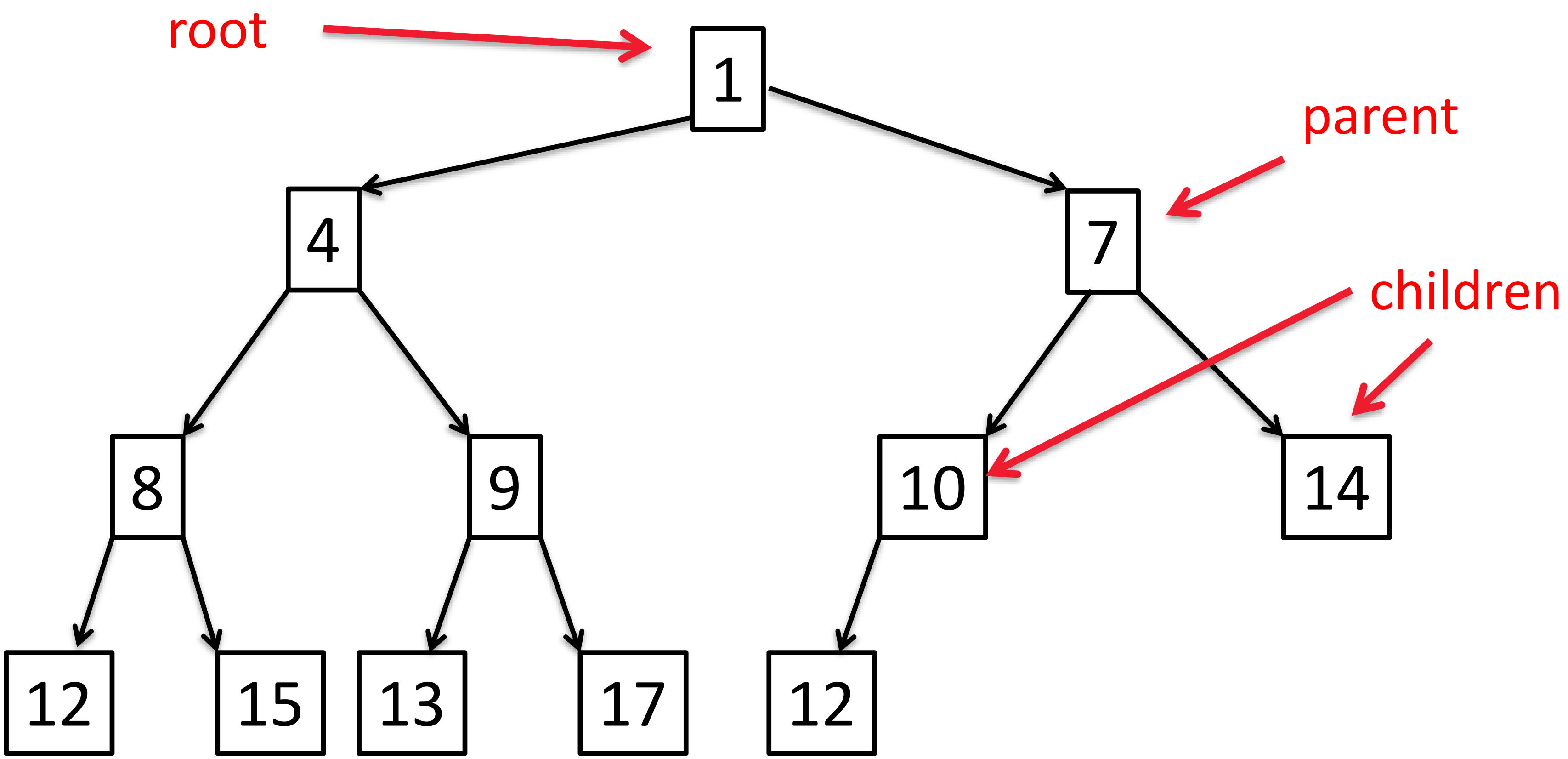
  e = min M;
  M \{e};
  return e

Heaps

- Tree-based data structure efficient implementation of priority queues

- Binary heap: the underlying tree is binary tree

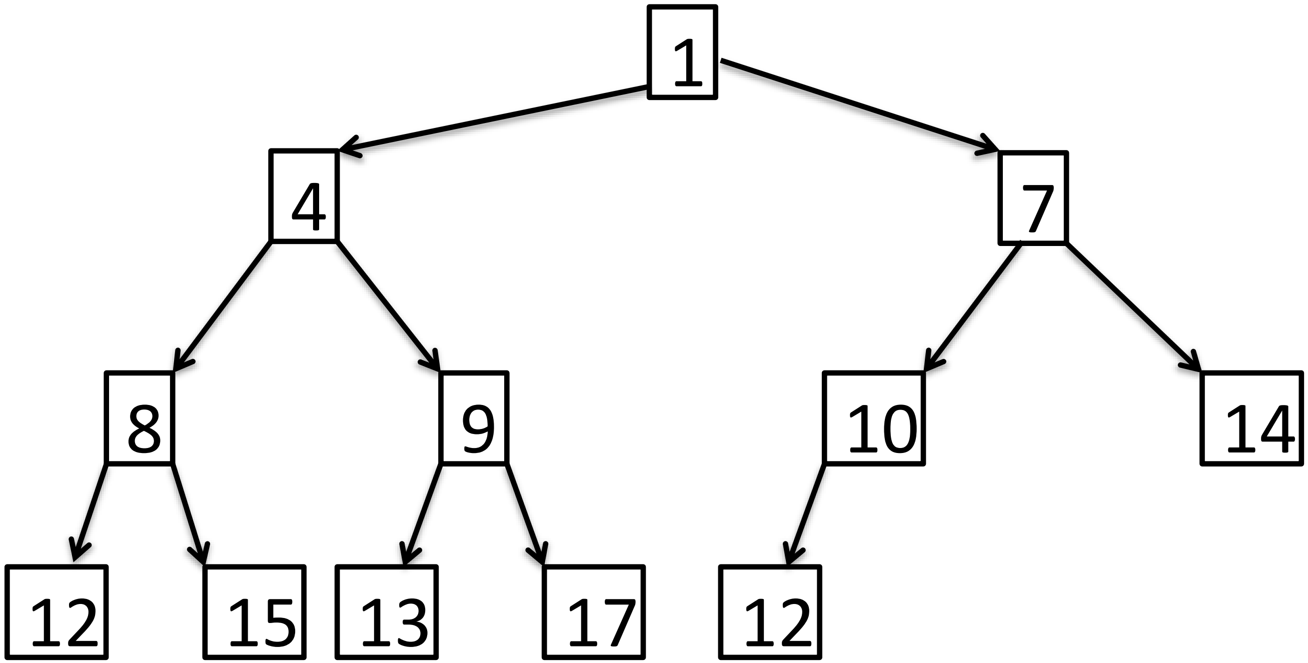
Binary heaps



Binary heaps – properties

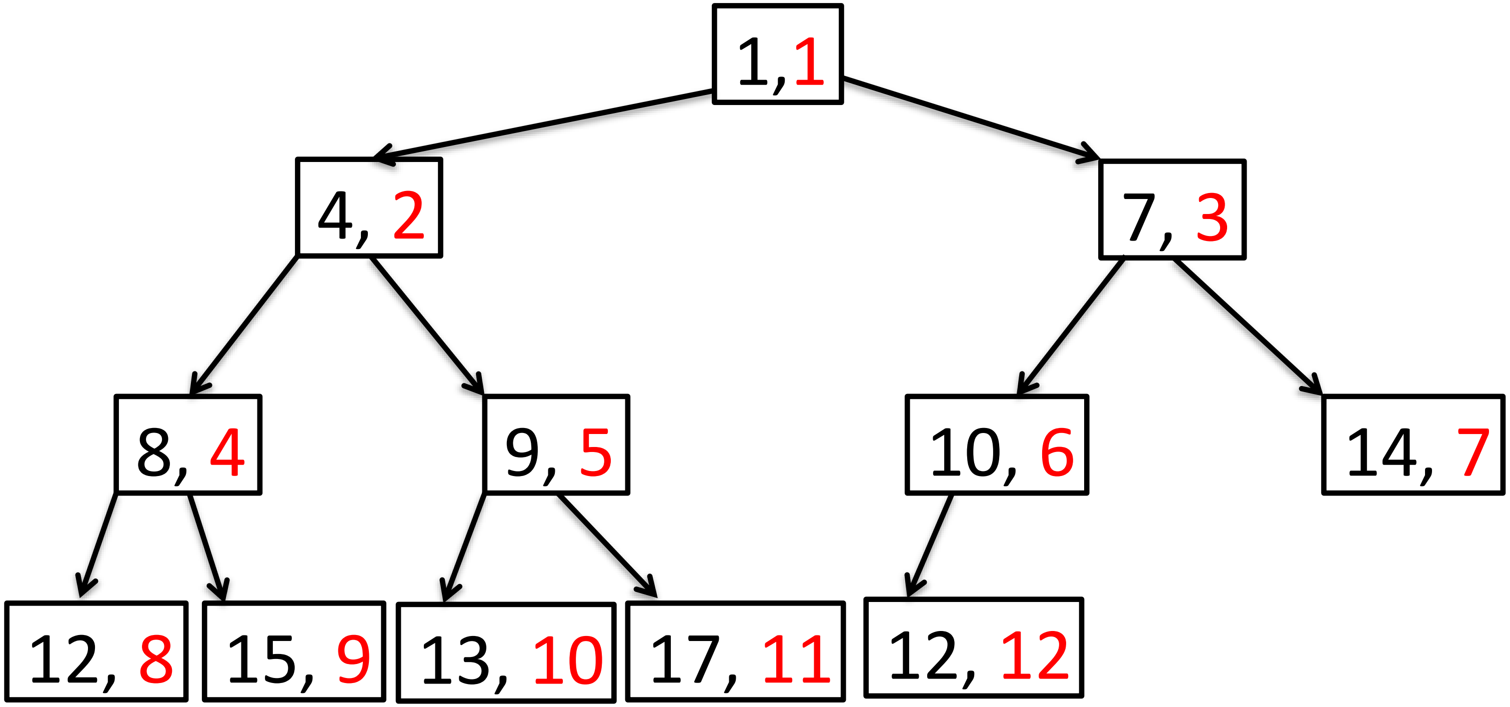
- Smallest element is stored at the root (min heap)
- Each node is smaller than its children (min heap)
- All levels are completely filled (except the last one)
- Last level is filled from left to right

Binary heaps as arrays



h:	1	4	7	8	9	10	14	12	15	13	17	12
indices:	1	2	3	4	5	6	7	8	9	10	11	12

Binary heaps as arrays



Observations

- The children of a node with index j have indices 2j and 2j+1 (if they exist)

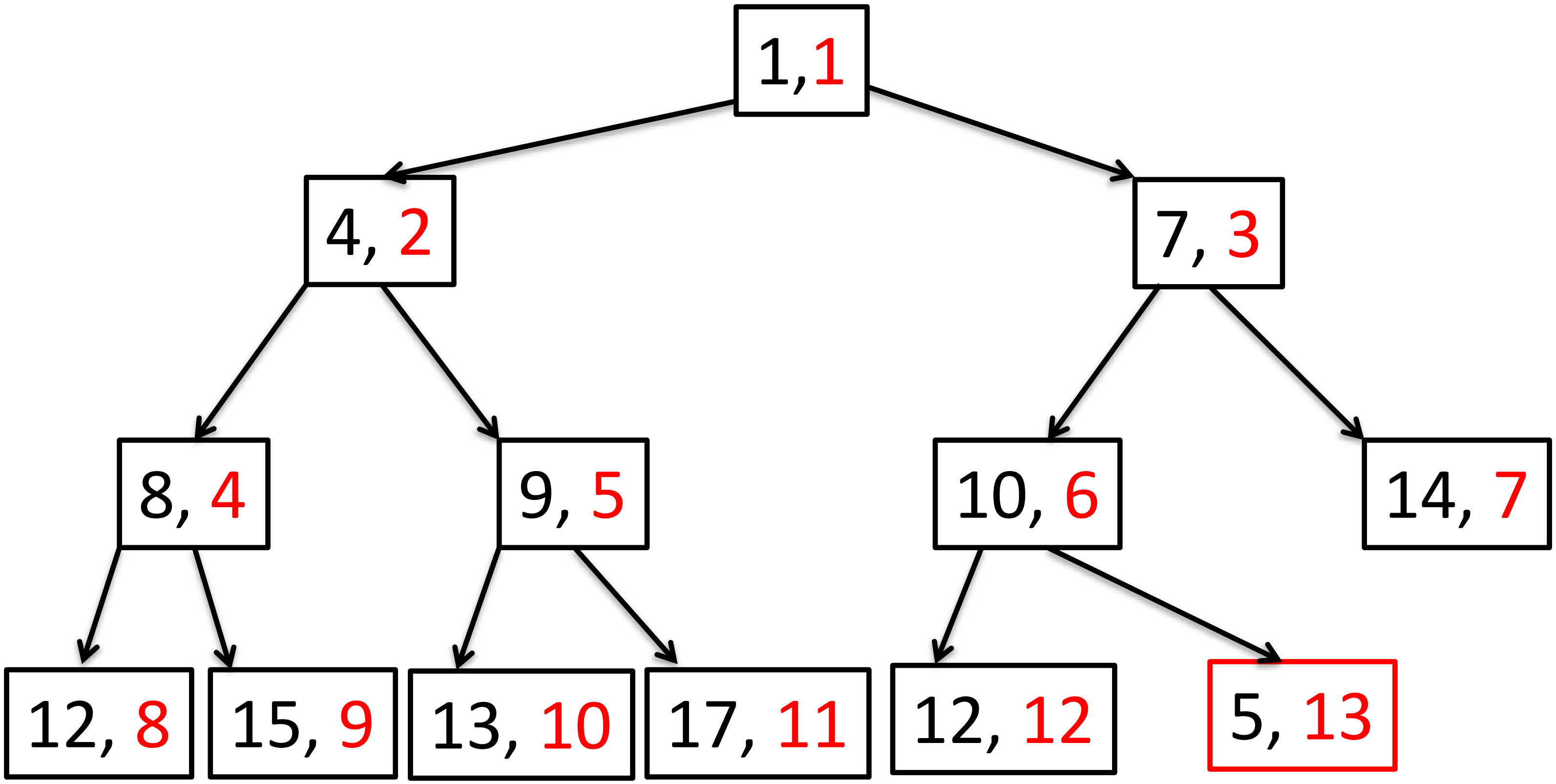
- Heap ordered: An array h is called heap-ordered iff

$$\forall j \in \{2, \ldots, n\} : h[\lfloor j/2 \rfloor] \leq h[j]$$

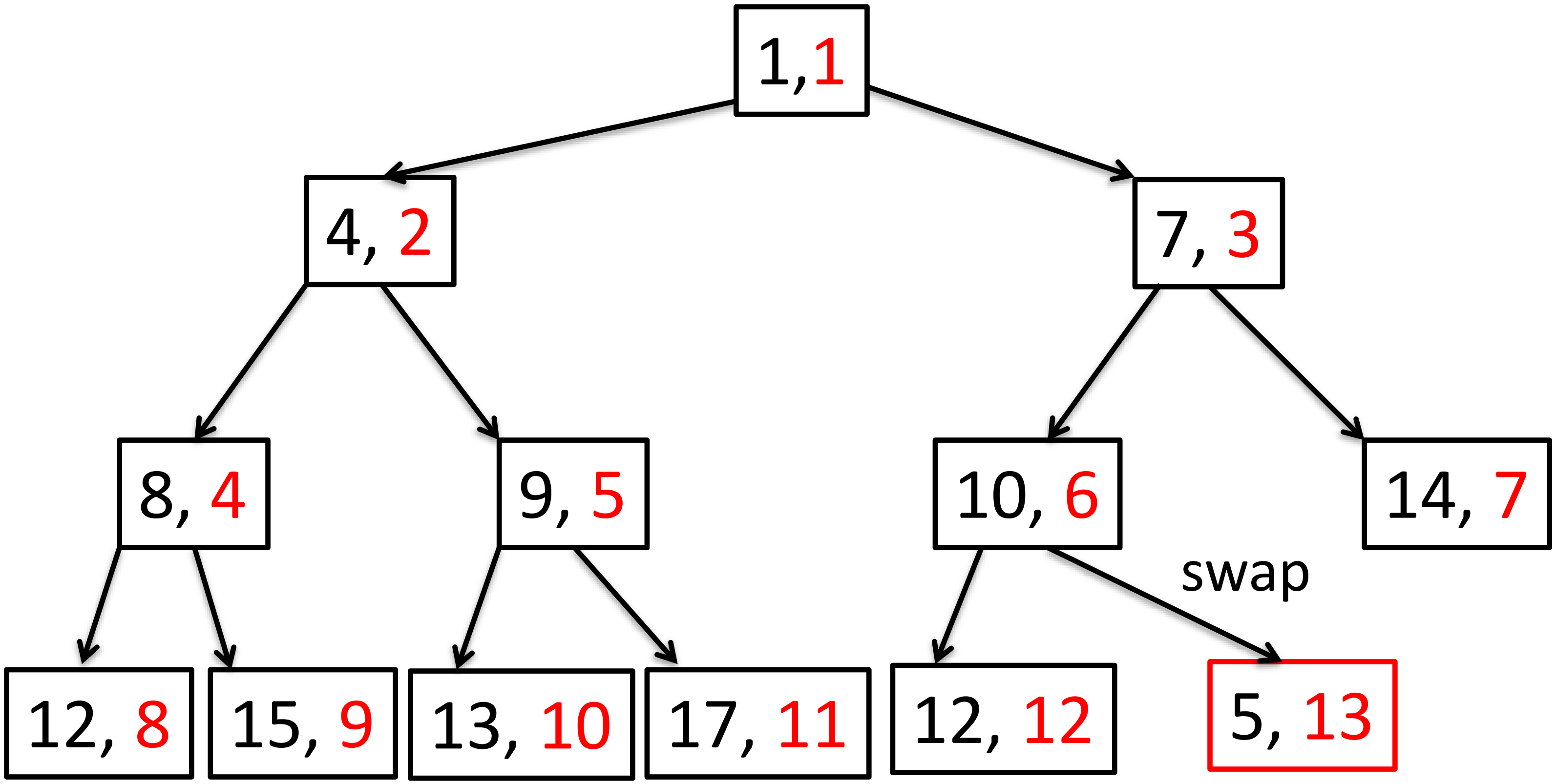
Insertion

```
insert(e)
        assert n< w          // heap not full (w = capacity)
        n++                  // update # of elements
        h[n] :=e             // insert e to heap array h
        siftUp(n)            // ensure heap property
```

Example insert

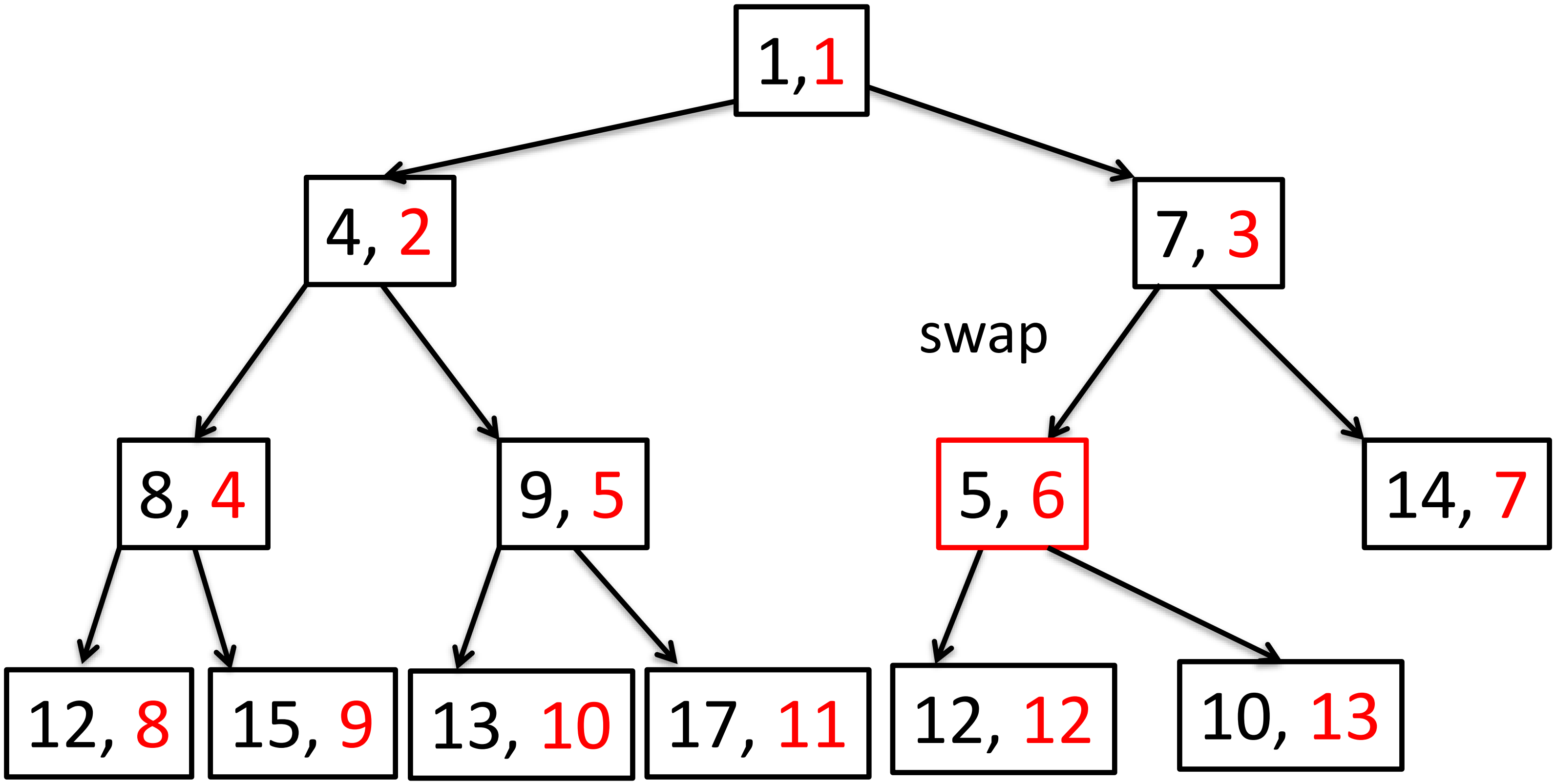


Example insert

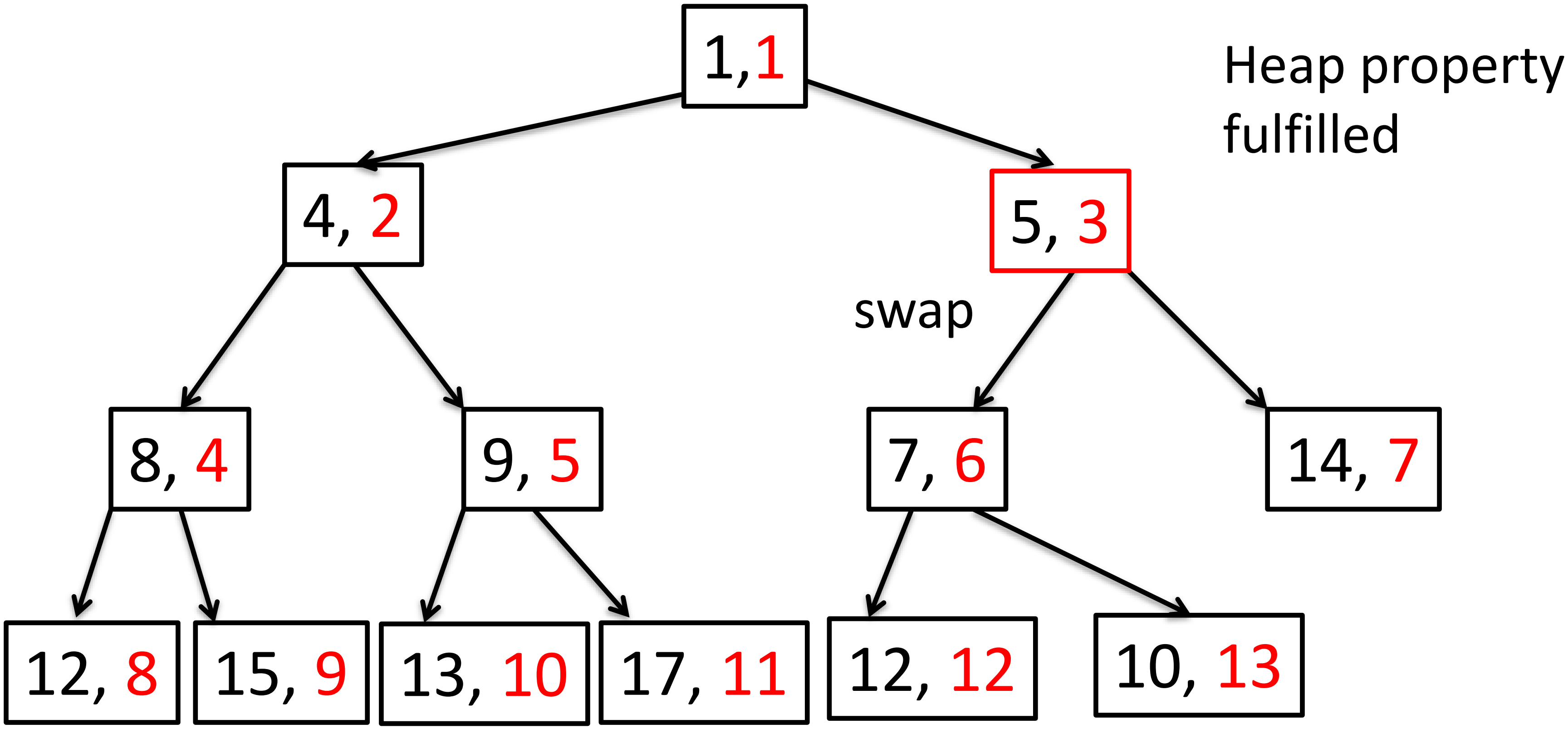


Heap property violated

Example – siftUp



Example – siftUp



SiftUp

siftUp(i)

assert heap property holds except maybe at position i

if $i = 1$ or $h[\lfloor i/2 \rfloor] \leq h[i]$ **then return**

assert heap property holds except for position i

$swap(h[i], h[\lfloor i/2 \rfloor])$

assert heap property holds except maybe at position $\lfloor i/2 \rfloor$

$siftUp(\lfloor i/2 \rfloor)$

Correctness of insertion

Assume: Heap property is fulfilled before call of insertion.

- Insertion calls sift up of element e and explores one path to the root.
- Heap property can only be violated at element e.
- Element e is moved up along the path until heap property on that path is established again.
- Heap property holds after insertion.

Deletion

deleteMin

```
assert n>0            // heap not empty
result= h[1]          // min value
h[1]:=h[n];           // remove value
n--                   // update # of elements
siftDown(1)           // ensure heap property
return result         // return min
```

SiftDown

siftDown(i)

assert heap property holds for trees rooted at *j=2i* and *j=2i+1*

if $2i \le n$ **then**

if $2i + 1 > n$ or $h[2i] \le h[2i+1]$ **then** $m := 2i$

else $m := 2i + 1$

assert sibling of m does not exist or its key is larger then m

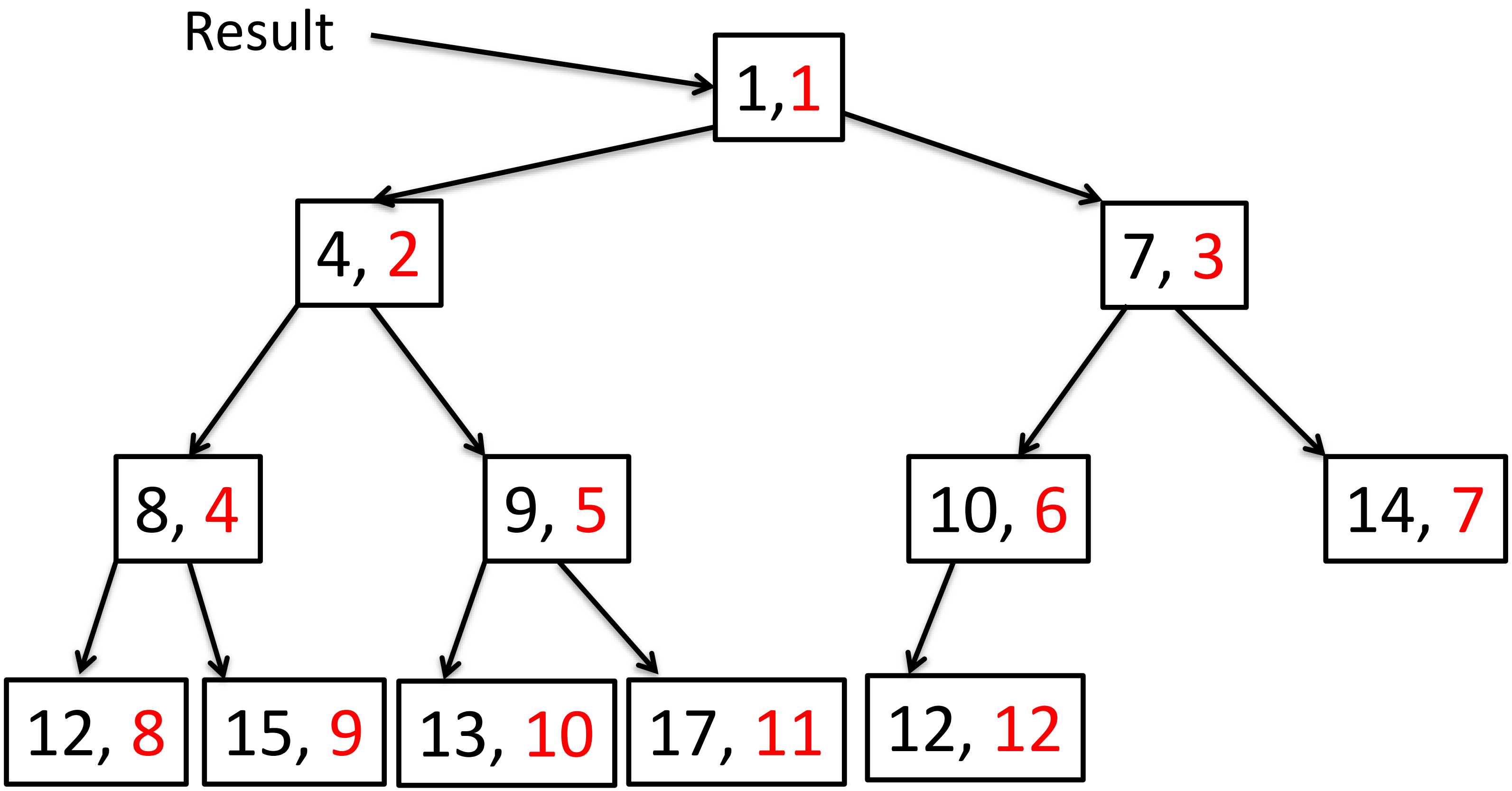
if $h[i] > h[m]$ **then**

$swap(h[i], h[m])$

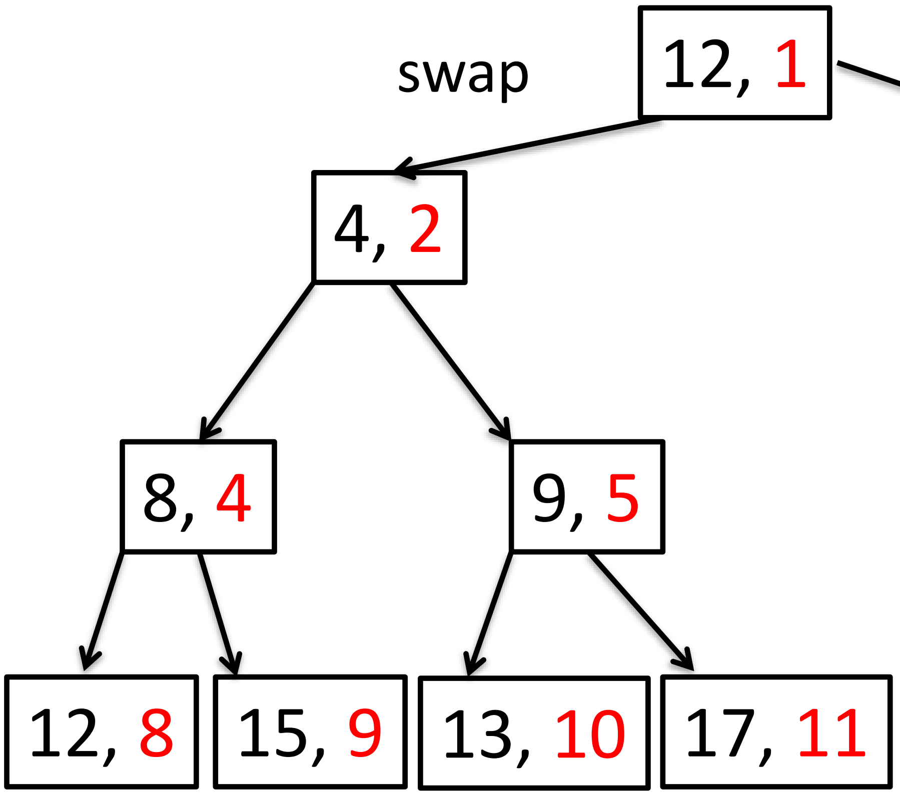
$siftDown(m)$

assert heap property holds for tree rooted at i

Example – deletionMin

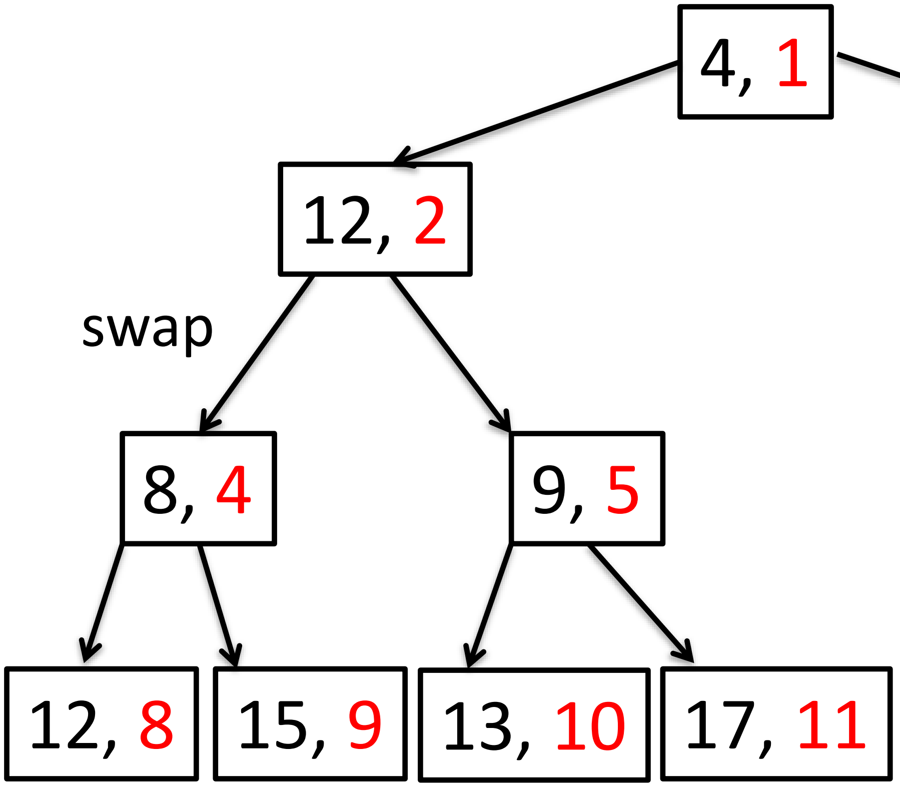


Example – deletionMin



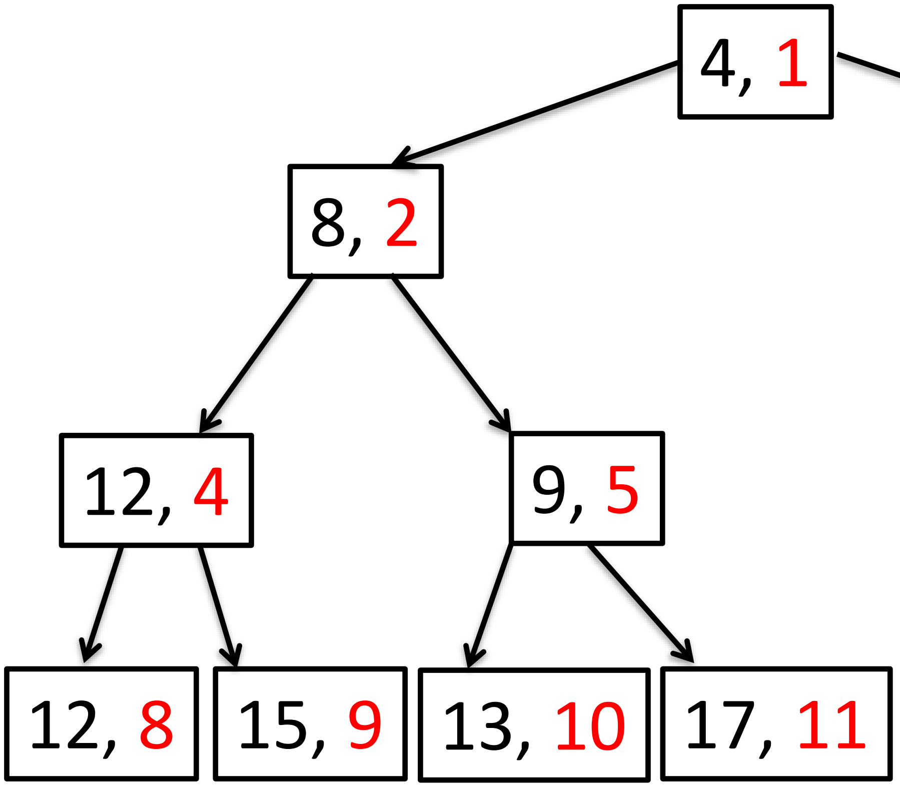
Heap property violated

Example – deletionMin



Heap property violated

Example – deletionMin



Heap property fullfilled

Correctness of deletionMin

Assume: Heap property is fulfilled before call of deletion.

- Deleting the first element and replacing it by the last element *e* in the array can violate the heap property at the children of the root.
- Heap property still holds at all other nodes.
- Deletion calls siftDown and explores one path from the root to a leaf.
- Heap property can only be violated at the children of *e*.
- SiftDown swaps *e* with the smallest of its children (implies heap property holds at the other child after the swap)
- Element *e* is moved down along the path until heap property is not violated at the children anymore.
- Heap property holds after deletion.

Theorem

- Insert and deleteMin have O(log n) time complexity

Proof sketch:

- Binary heap of n element has height $k = \lfloor \log n \rfloor$
- Insert and deleteMin explore one root-to-leaf path
- Hence have logarithmic running time

$\square$