Heap sort

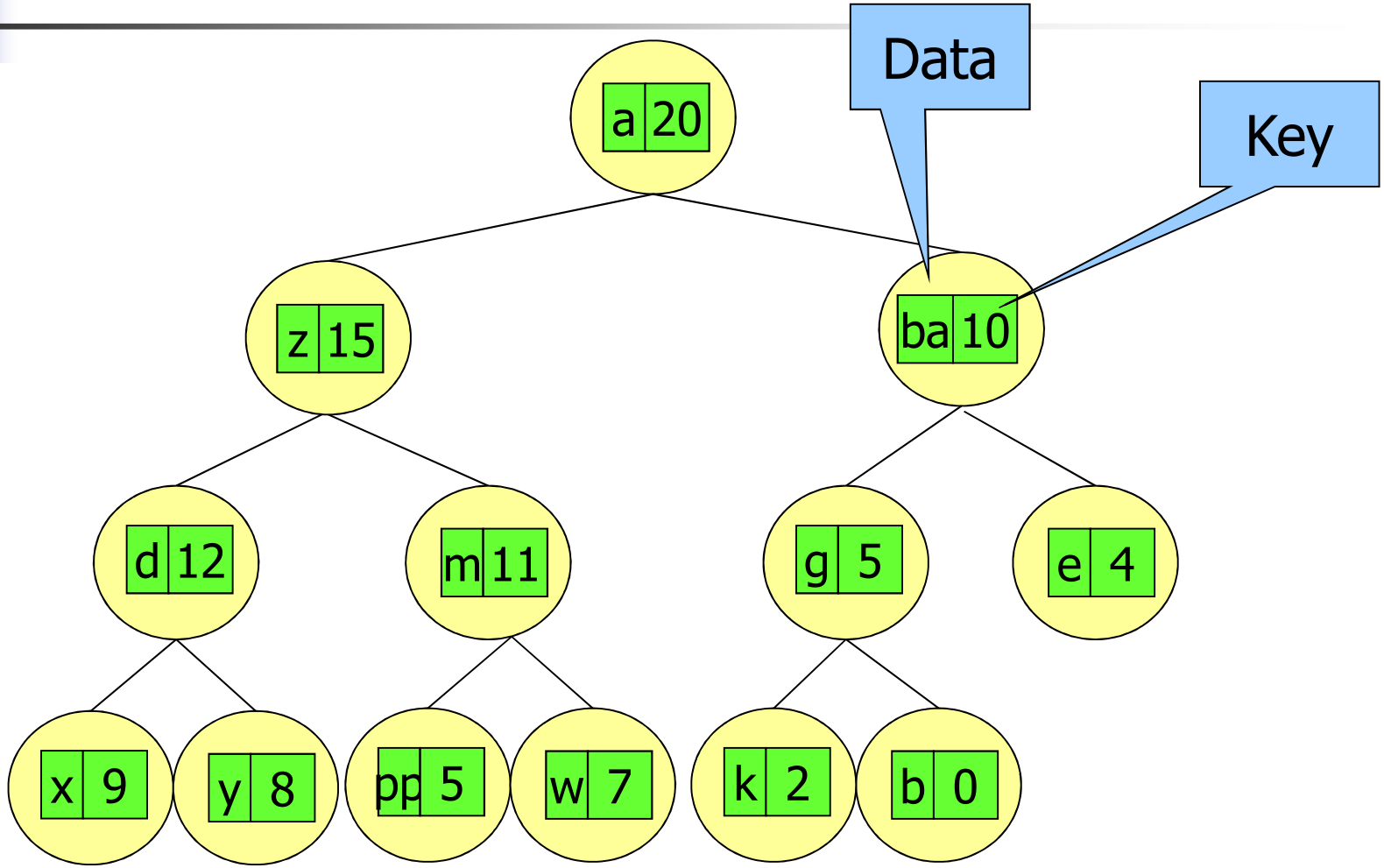
Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

ADT Heap

- A heap is a binary tree with
  - A structural property
    - Almost complete (all levels are complete, possibly except the last one, filled from left to right), i.e., the tree is almost balanced
  - A functional property
    - $\forall$ node $\neq$ root we have that

      key(parent(node)) $\geq$ key(node)
- Consequence
  - The maximum key is in the root

Example: A heap



ADT Heap

- Data structure

```
struct heap {
  Item *A;
  int heapsize;
};
```

Array A[0..maxN-1] of Item

Number of elements in heap h->A

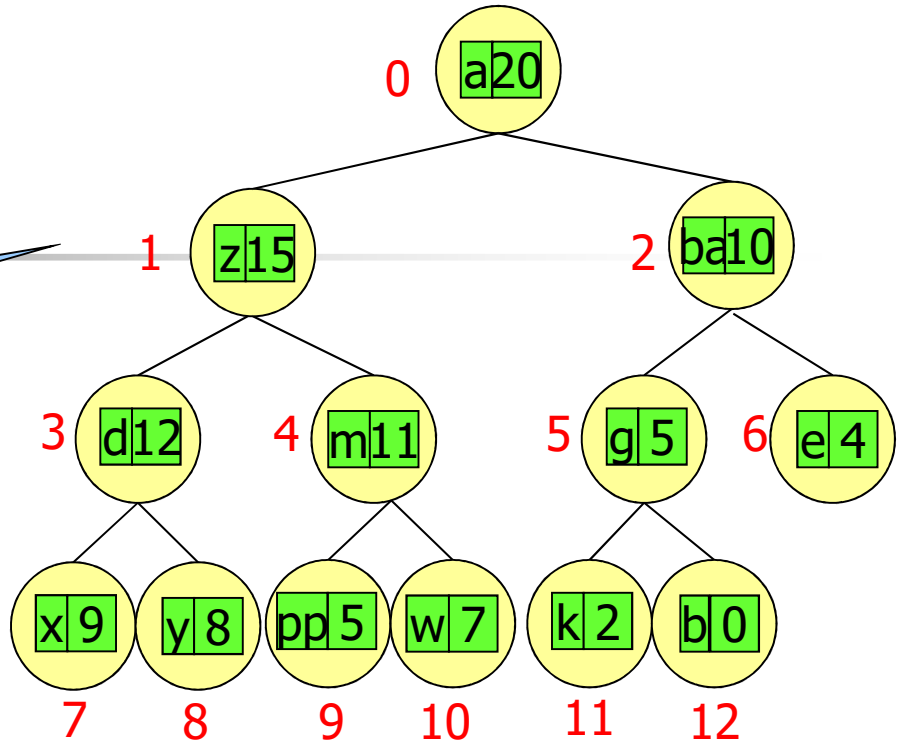
- Operations
  - HEAPify(h,i)
  - HEAPbuild(h)
  - HEAPsort(h)

Implementation details

- The root is stored in h->A[0]
- Given h->A[i]
  - The left child is h->A[LEFT(i)]
    - Where LEFT(i)= 2i+1
  - The right child is h->A[RIGHT(i)]
    - Where RIGHT(i) = 2i+2
  - The parent is h->A[PARENT(i)]
    - Where PARENT(i)=(i-1)/2

Example

Heap



Array implementation

h->A

a	z	ba	d	m	g	e	x	y	pp	w	k	b		
20	15	10	12	11	5	4	9	8	5	7	2	0		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

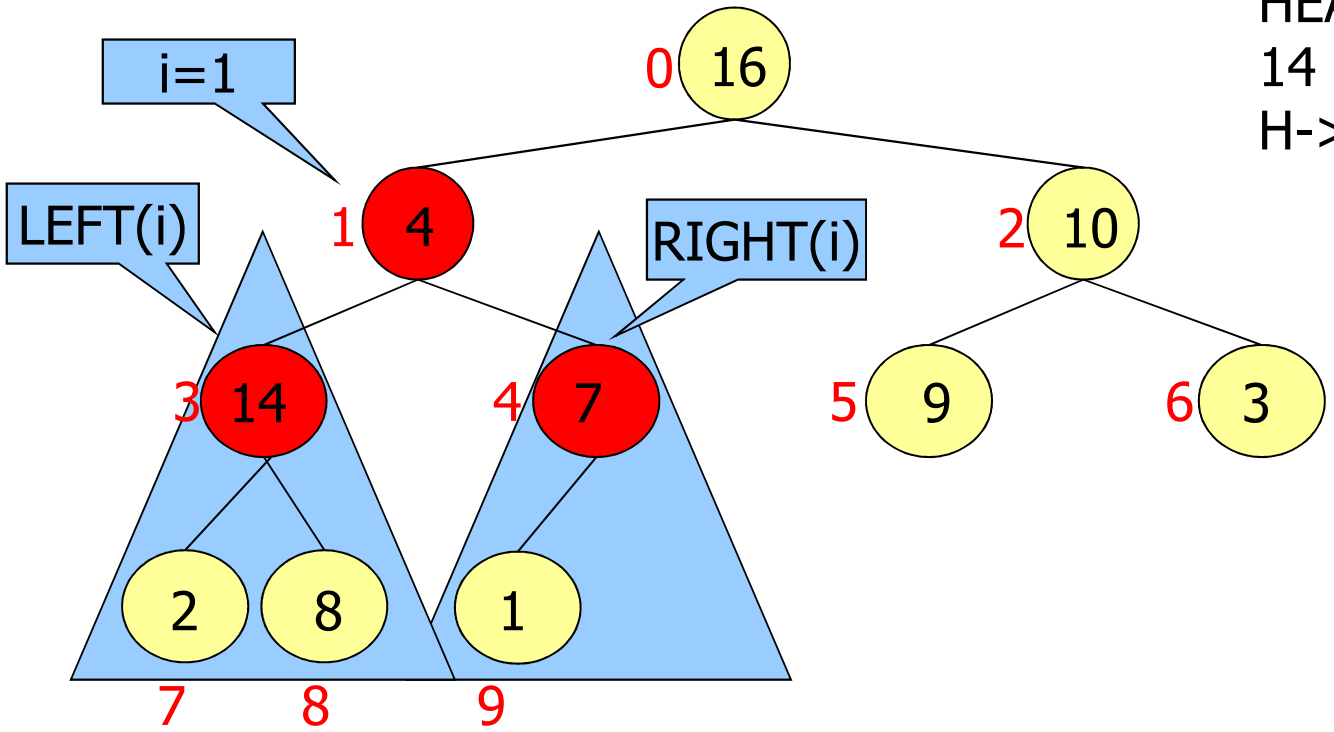
h->heapsize 13

maxN = 15

Function HEAPify

- Sub-trees LEFT(i) and RIGHT(i) are already heaps
- Turns into a heap tree i, LEFT(i), RIGHT(i)
  - Assigns to A[i] the maximum between A[i], LEFT(i) and RIGHT(i)
  - If there has been a swap A[i] $\leftrightarrow$ LEFT(i), recursively apply HEAPify on the subtree whose root is LEFT(i)
  - Dually in the case of a swap [i] $\leftrightarrow$ RIGHT(i)
- Complexity
  - $T(n) = O(\lg n)$

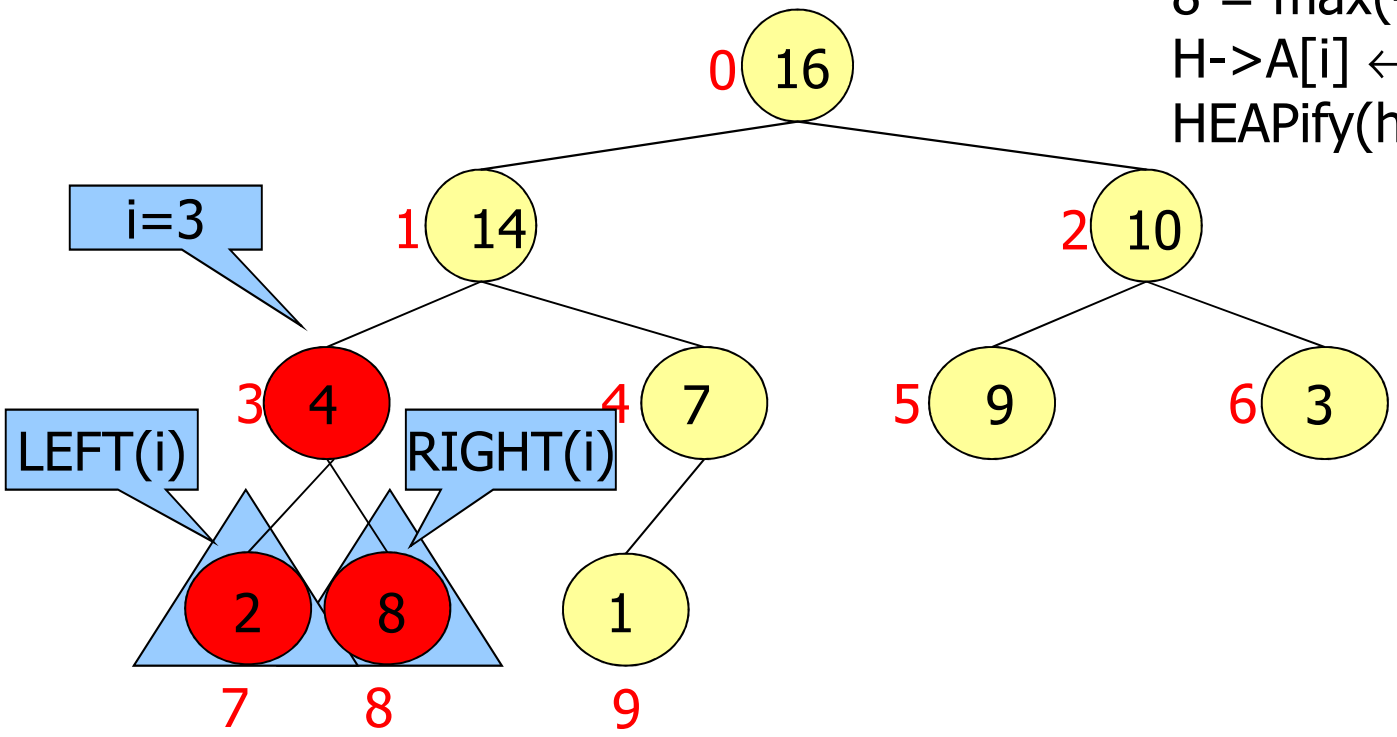
Example



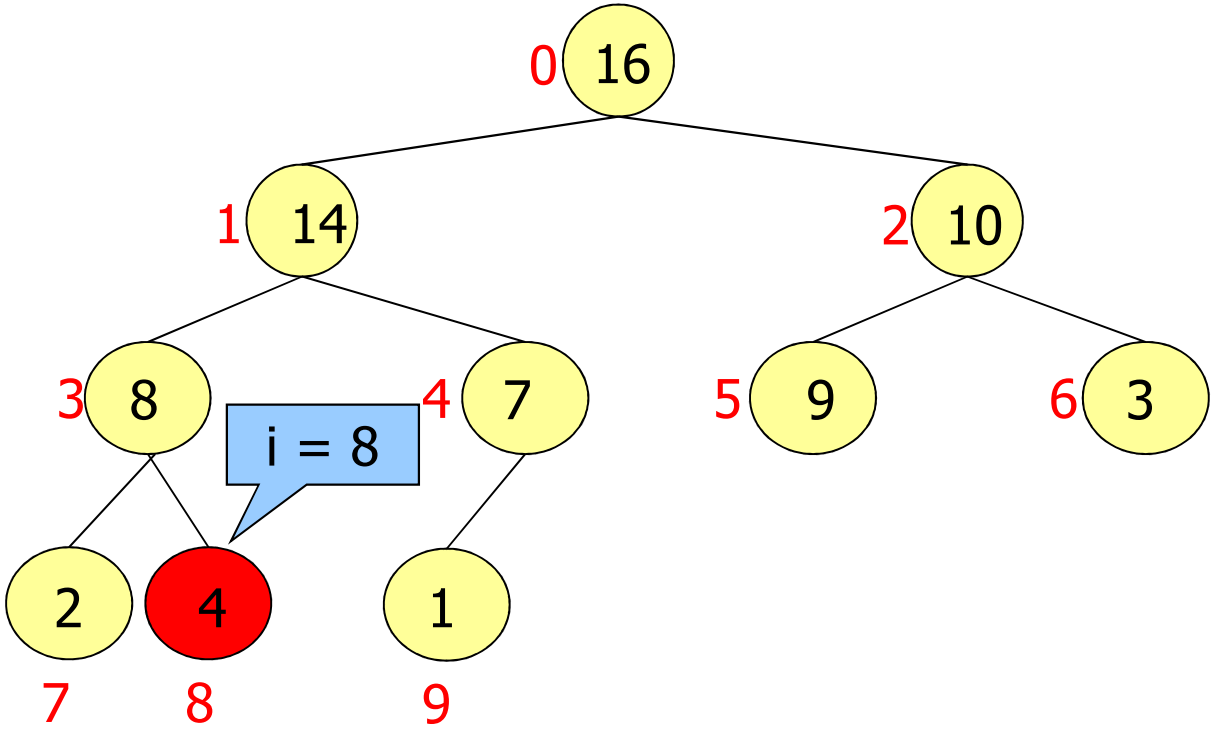
HEAPify(h->A, 1)
14 = max(4, 14, 7)
H->A[i] ↔ h->A[LEFT(i)]

Example

HEAPify(h->A, 3)
8 = max(4, 2, 8)
H->A[i] ↔ h->A[RIGHT(i)]
HEAPify(h->A, 8)



Example



HEAPify(h->A, 8)
leaf
termination.

Implementation

```
void HEAPify(Heap h, int i) {
  int l, r, largest;
  l = LEFT(i);
  r = RIGHT(i);
  if ((l<h->heapsize) && (ITEMgreater(h->A[l], h->A[i])))
   largest = l;
  else
    largest = i;
  if((r<h->heapsize)&&(ITEMgreater(h->A[r],h->A[largest])))
    largest = r;
  if (largest != i) {
    Swap(h, i,largest);
    HEAPify(h, largest);
  }
  return;
}
```

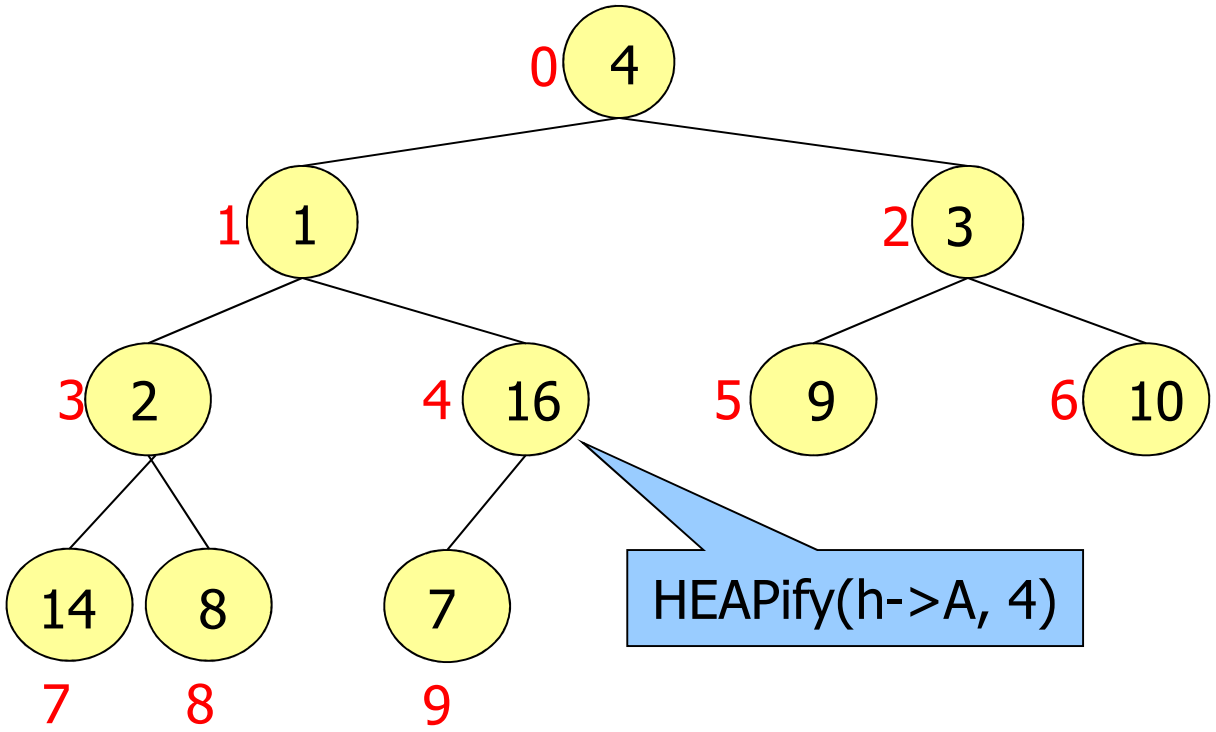
Function HEAPbuild

- Turns a binary tree stored in an array A into a heap
  - Leaves are heaps
  - Apply HEAPify starting from the parent node of the last pair of leaves back until the root
- Complexity
  - T(n)= O(n)

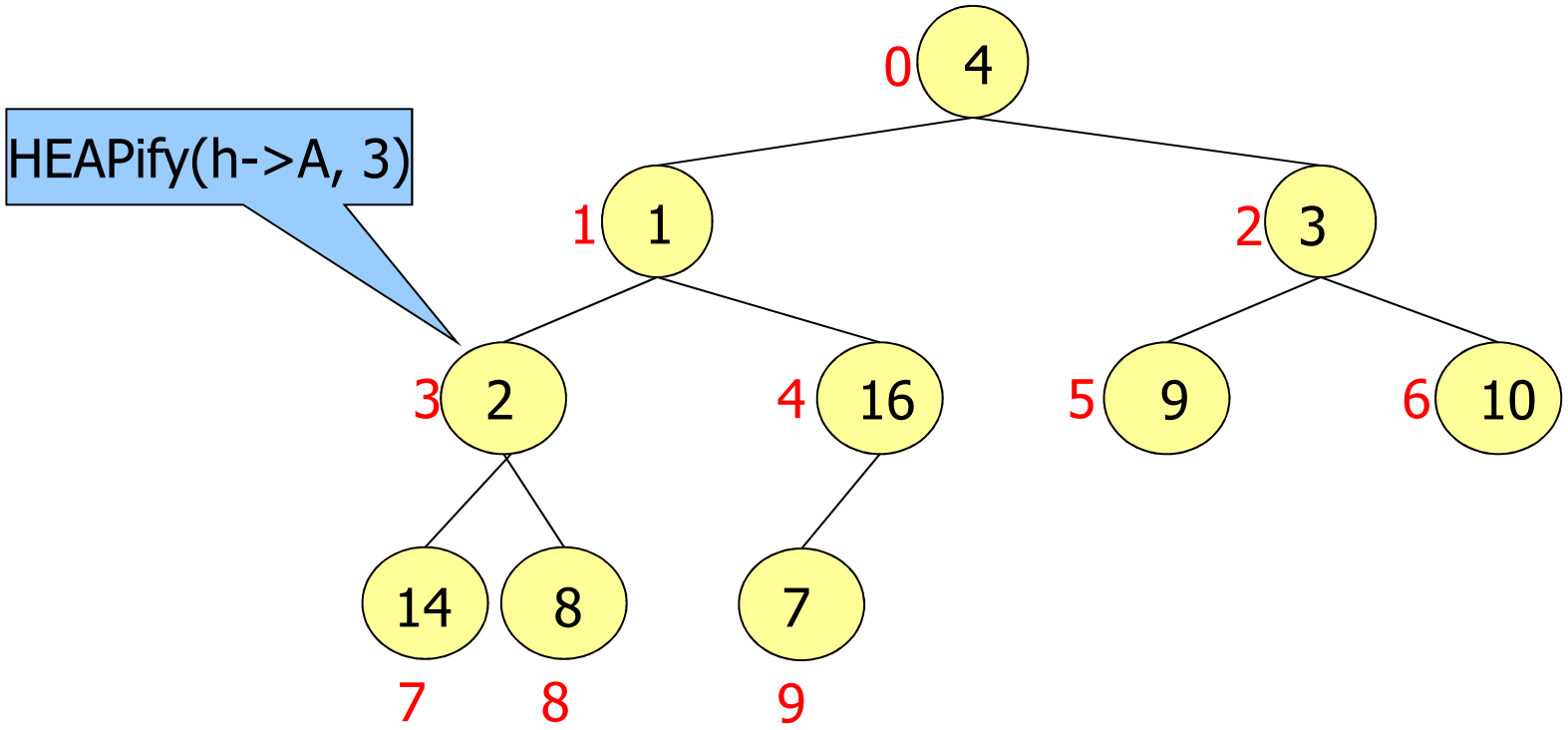
Example

Only integer keys are shown, not items

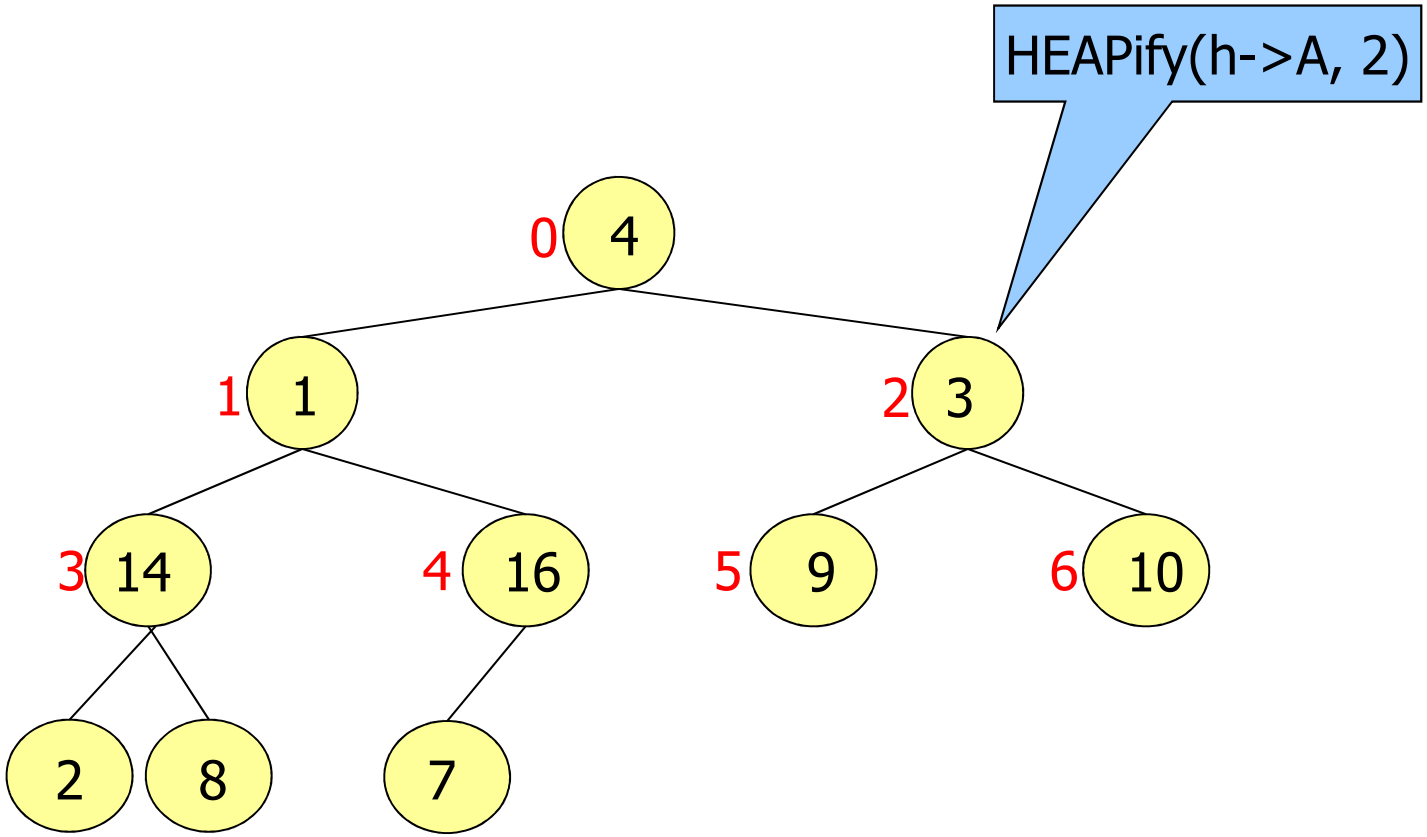
HEAPbuild(h->A)



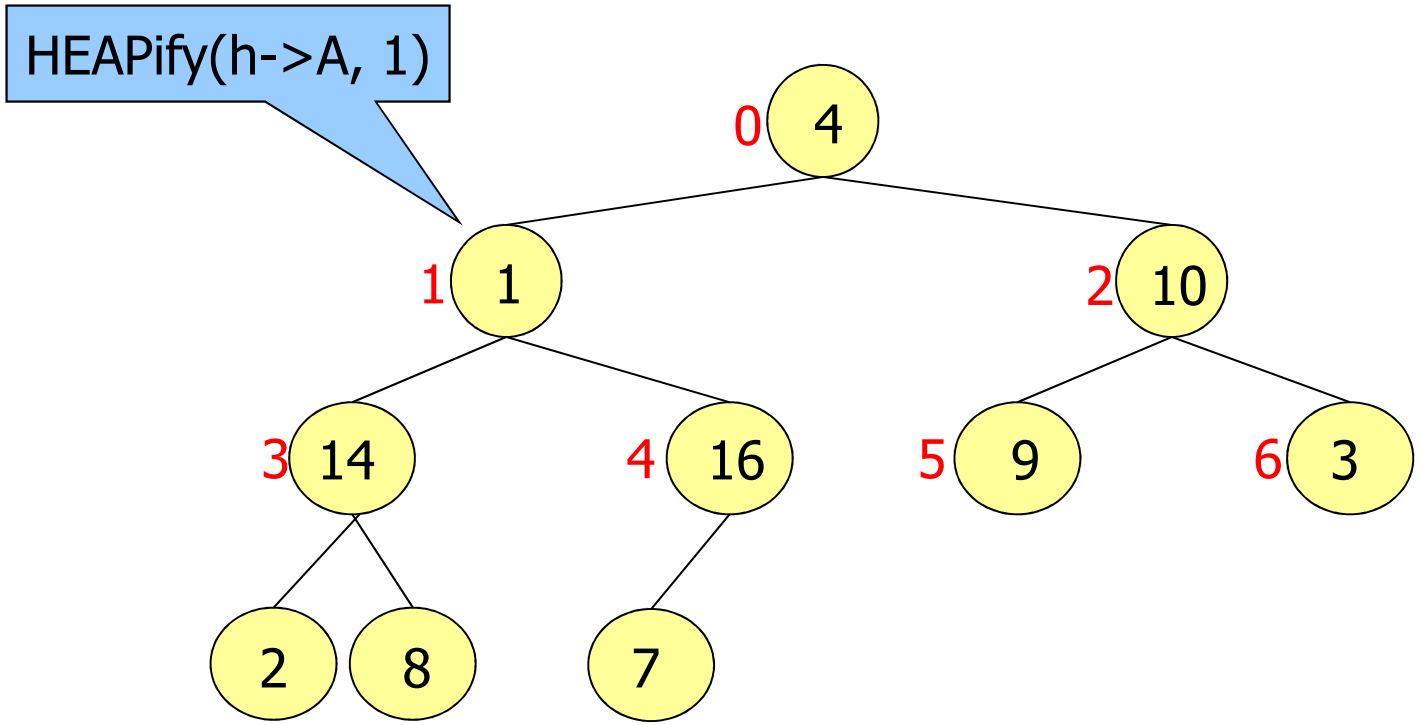
Example



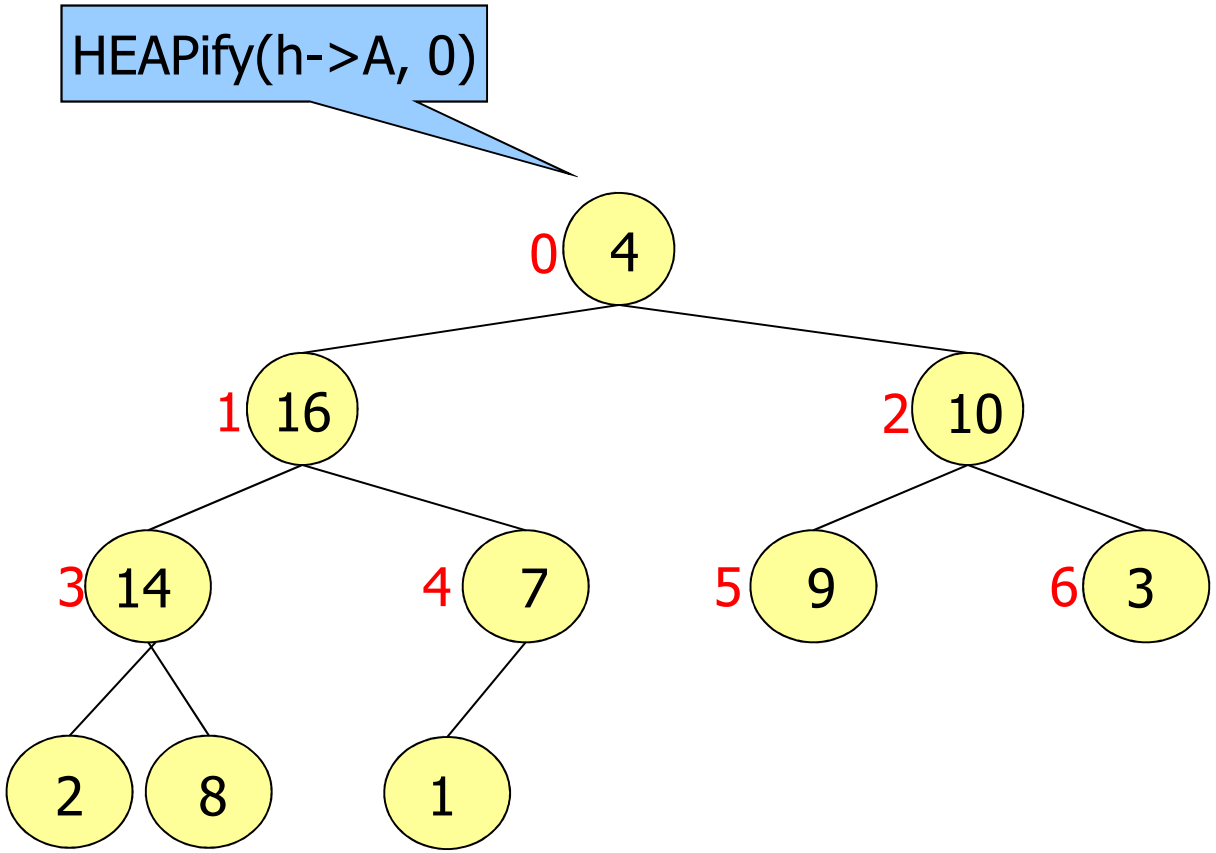
Example



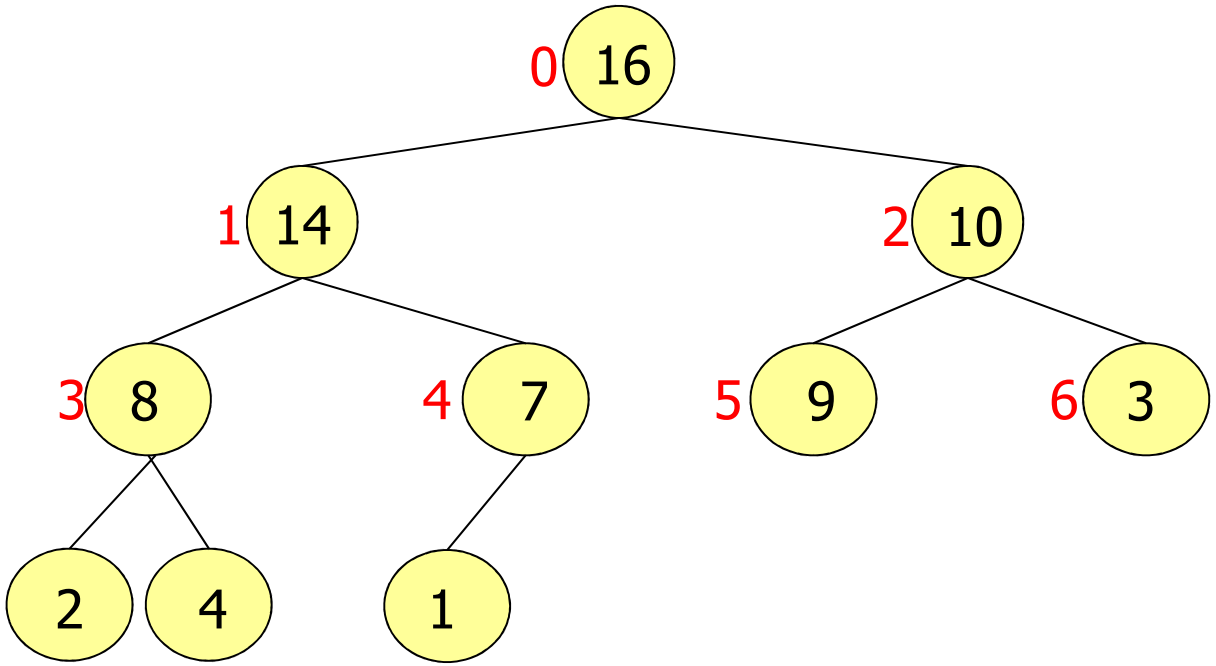
Example



Example



Example



Implementation

```
void HEAPbuild (Heap h) {
  int i;

  for (i=(h->heapsize)/2-1; i >= 0; i--)
    HEAPify(h, i);

  return;
}
```

Function HEAPsort

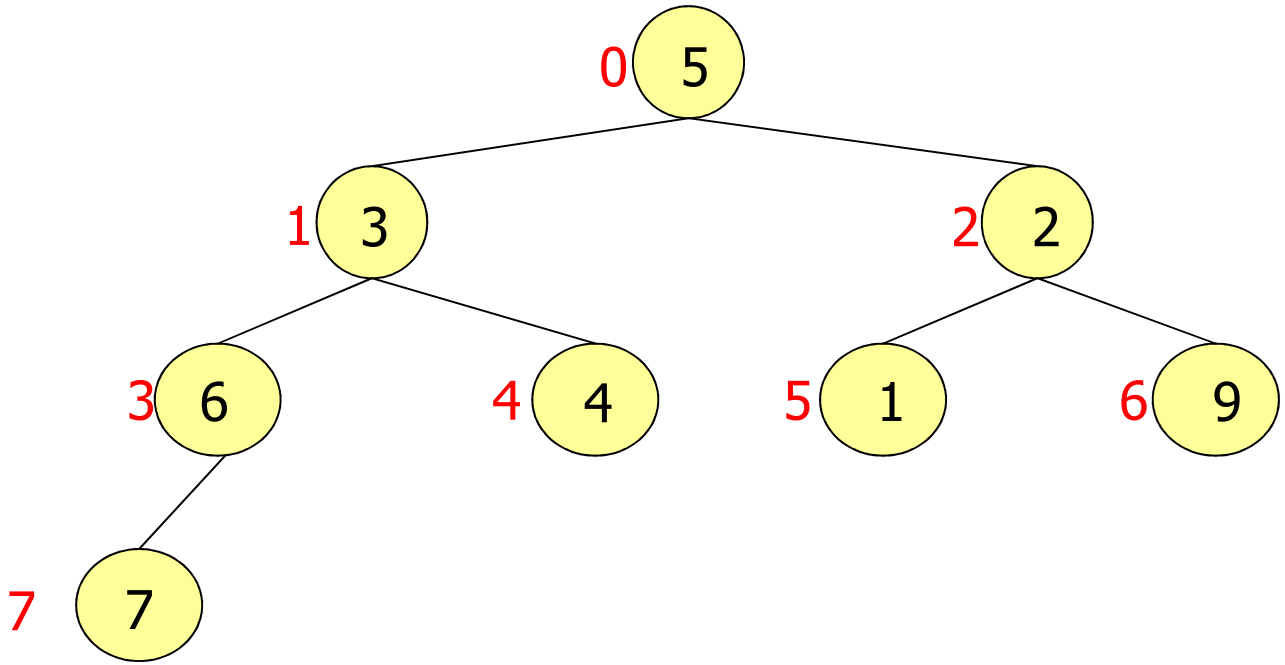
- Turns the array into a heap using HEAPbuild
  - Swaps first and last elements
  - Decreases heap size by 1
  - Reinforces the heap property heap
  - Repetat until heap empty
- Complexity
  - $T(n)= O(n \lg n)$
- In place
- Not stable

Example

Only integer keys are shown, not items

Initial configuration

h->A	5	3	2	6	4	1	9	7

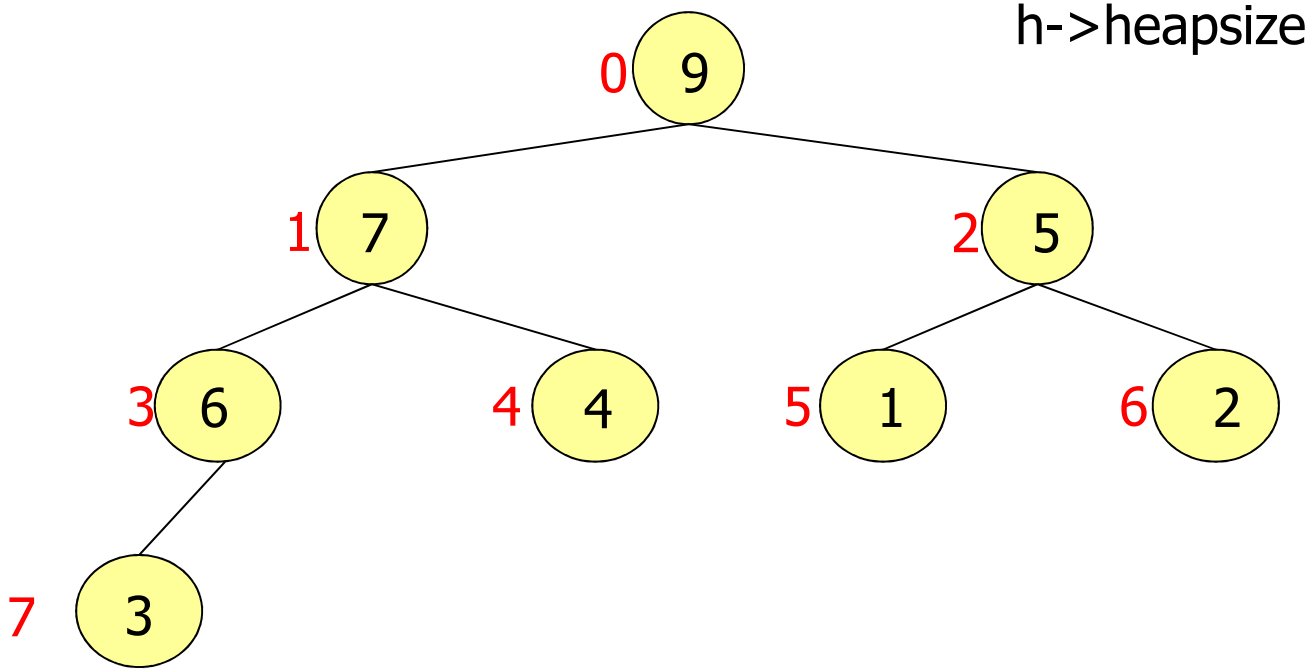


Example

Apply HEAPbuild

h->A	9	7	5	6	4	1	2	3

h->heapsize = 8



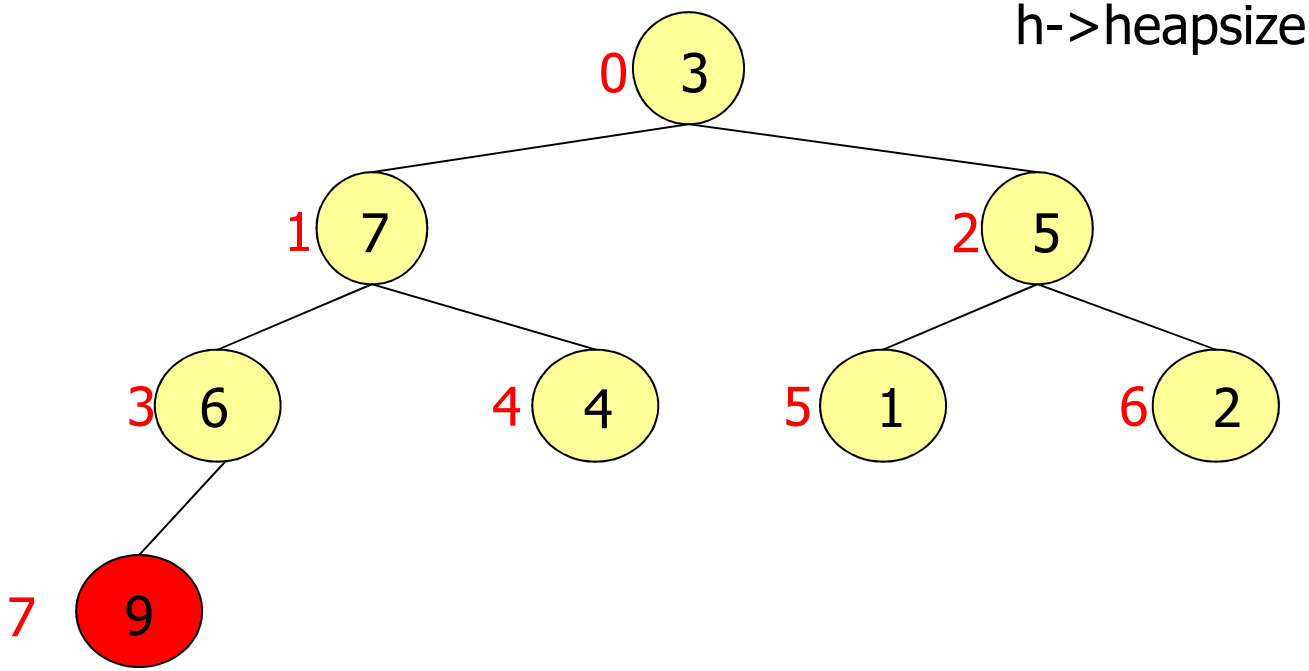
Example

h->A[0] ↔ h->A[h->heapsize-1]
h->heapsize--

h->A

3	7	5	6	4	1	2	9

h->heapsize = 7



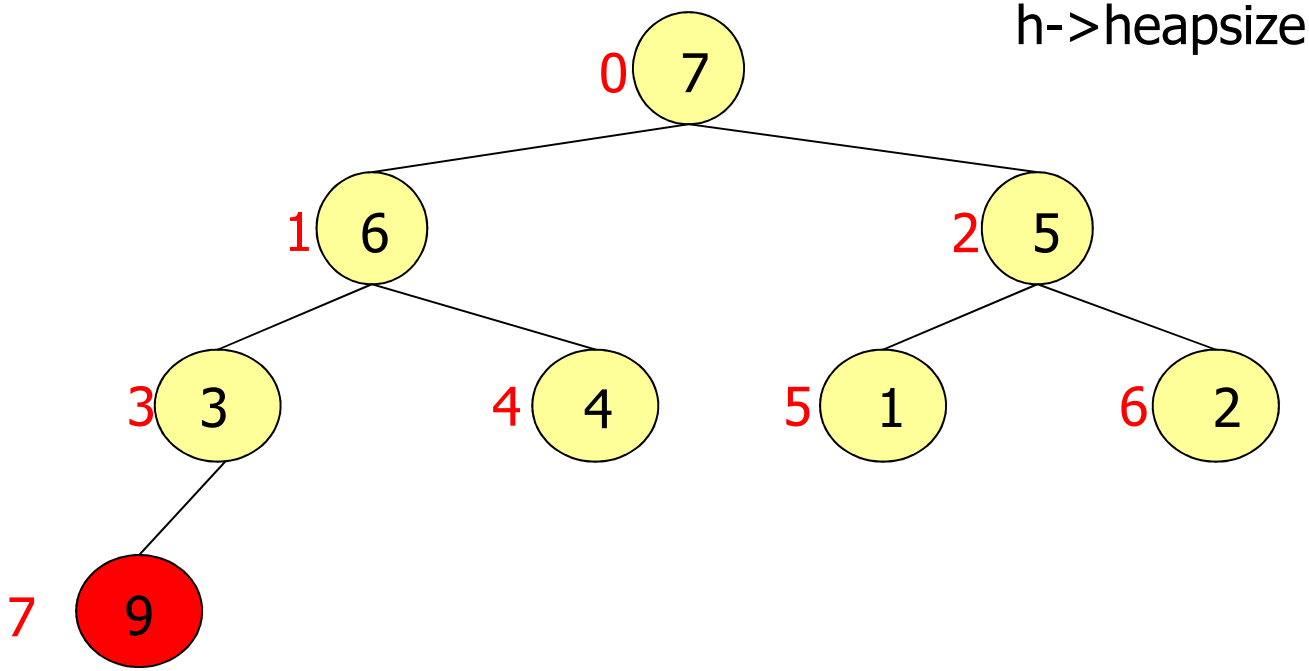
Example

HEAPify(h->A, 0)

h->A

7	6	5	3	4	1	2	9

h->heapsize = 7



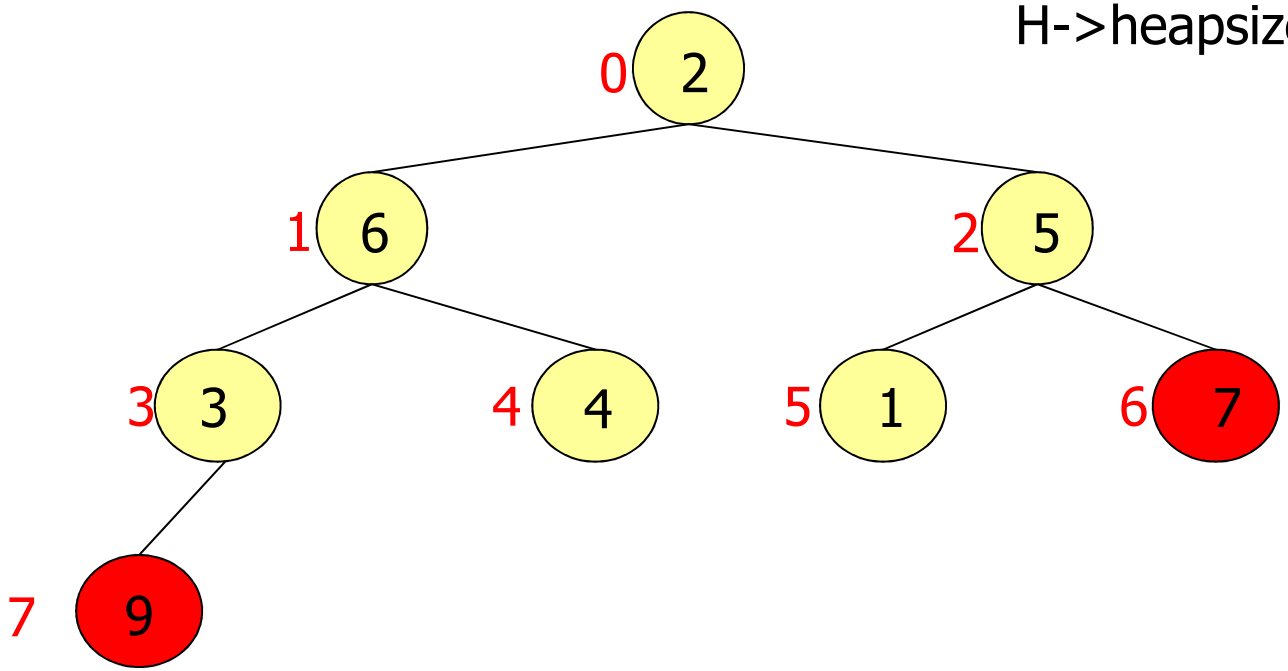
Example

h->A[0] ↔ h->A[h->heapsize-1]
h->heapsize--

h->A

2	6	5	3	4	1	7	9

H->heapsize = 6



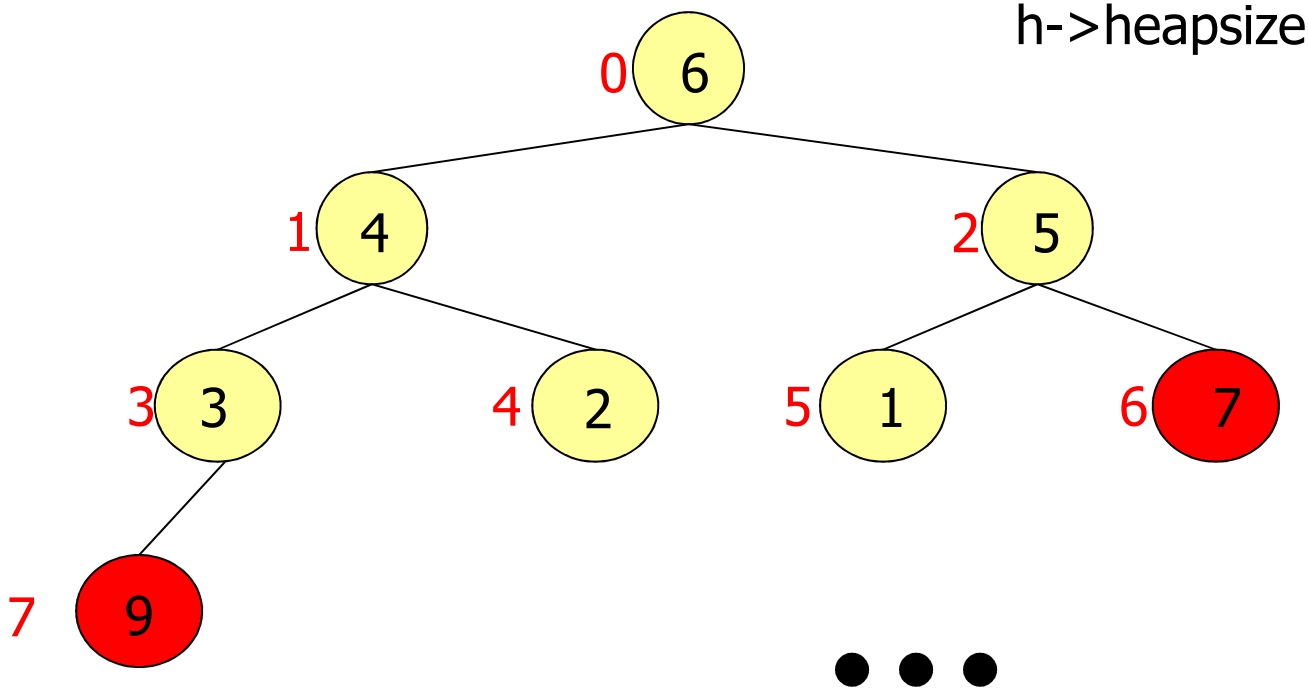
Example

HEAPify(h->A, 0)

h->A

6	4	5	3	2	1	7	9

h->heapsize = 6



• • •

Implementation

```
void HEAPsort(Heap h) {
  int i, j;
  HEAPbuild(h);
  j = h->heapsize;
  for (i = h->heapsize-1; i > 0; i--) {
    Swap (h,0,i);
    h->heapsize--;
    HEAPify(h,0);
  }
  h->heapsize = j;
  return;
}
```