HEAPS & HEAP SORT

Priority Queue

Implements a set S of elements, each of elements associated with a key.

INSERT(S, x): insert x into set S

MAX(S): return element of S with the largest key

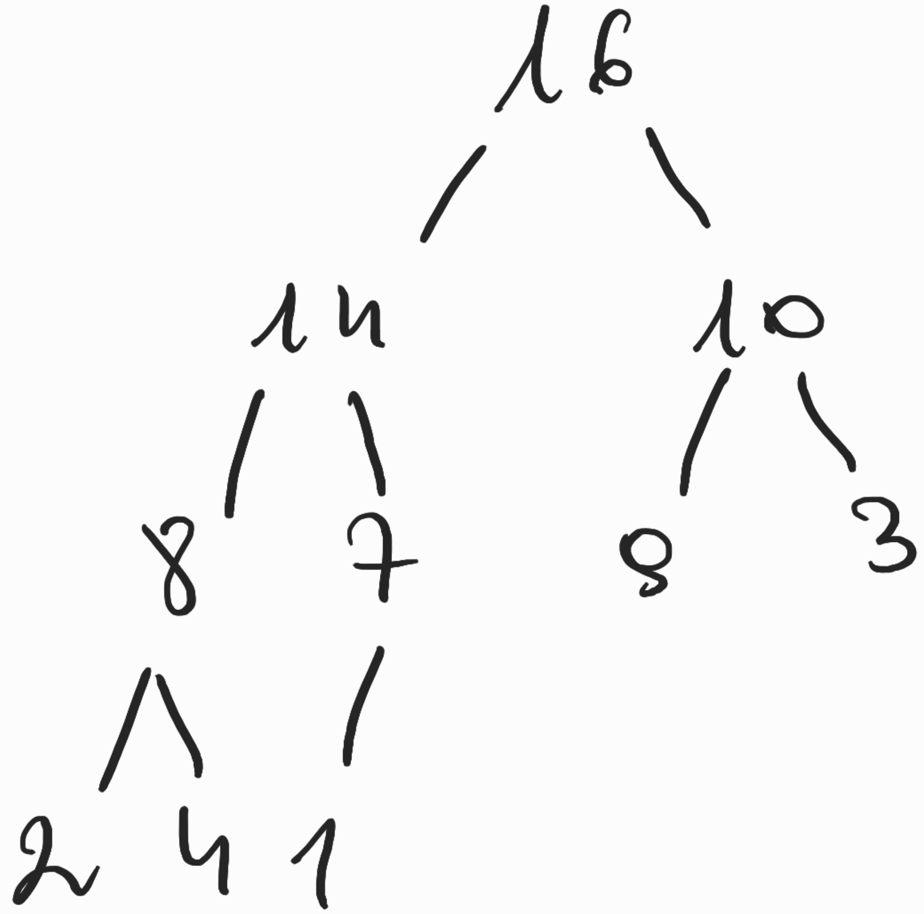
EXTRACT_MAX(S): do what MAX(S) do but also remove it from S

INCREASE_KEY(S, x, k): increase the value of x's key to new value k.

Heap

An array visualized as a nearly complete binary tree

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1



Heap as a Tree

root of tree: first element ($i=1$)

parent$(i) = i/2$

left child $= 2i$

right child $= 2i+1$

right child 2i+1

MAX-HEAP PROPERTY

The key of a node is $\geq$ the keys of its children

HEAP OPERATION

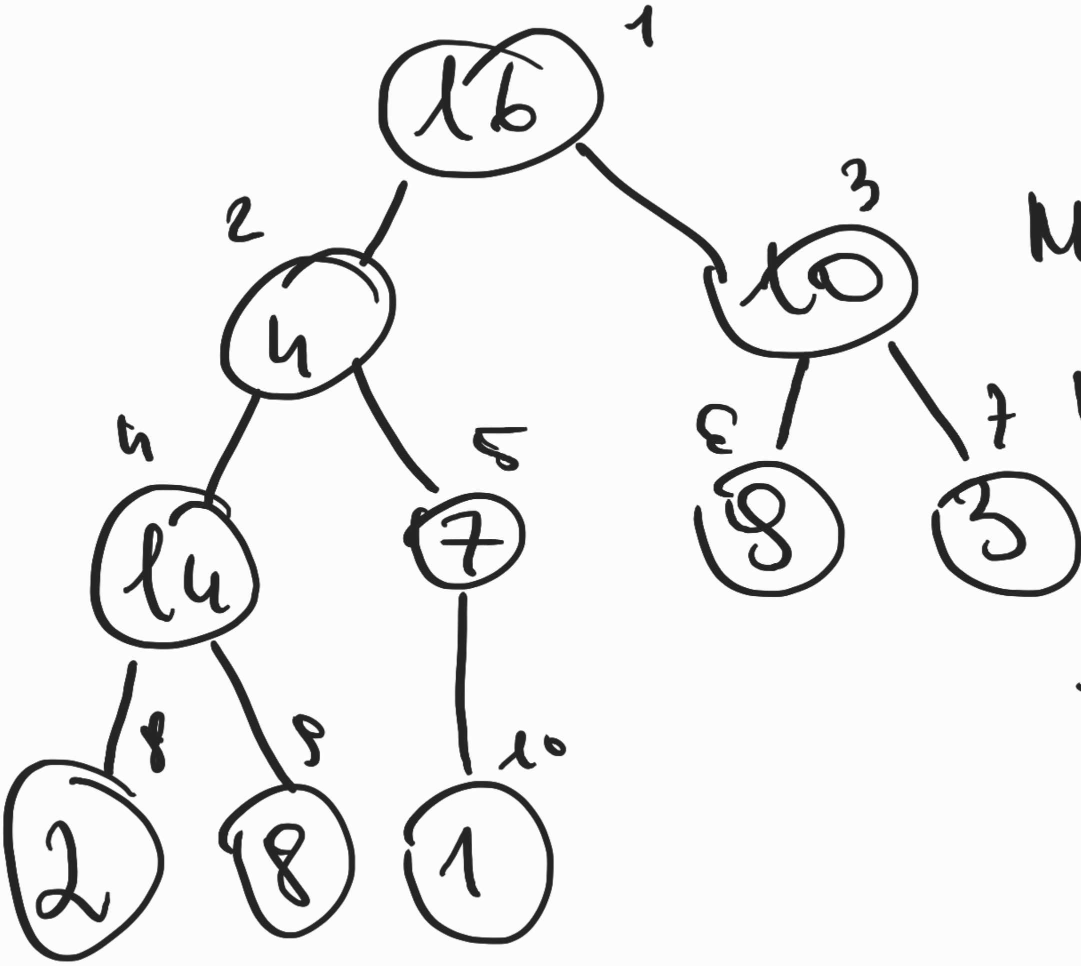
build_max_heap : produces a maxheap from an unordered array

max_heapify : correct a single violation of the heap property in a subtree's root

ASSUME THAT the trees

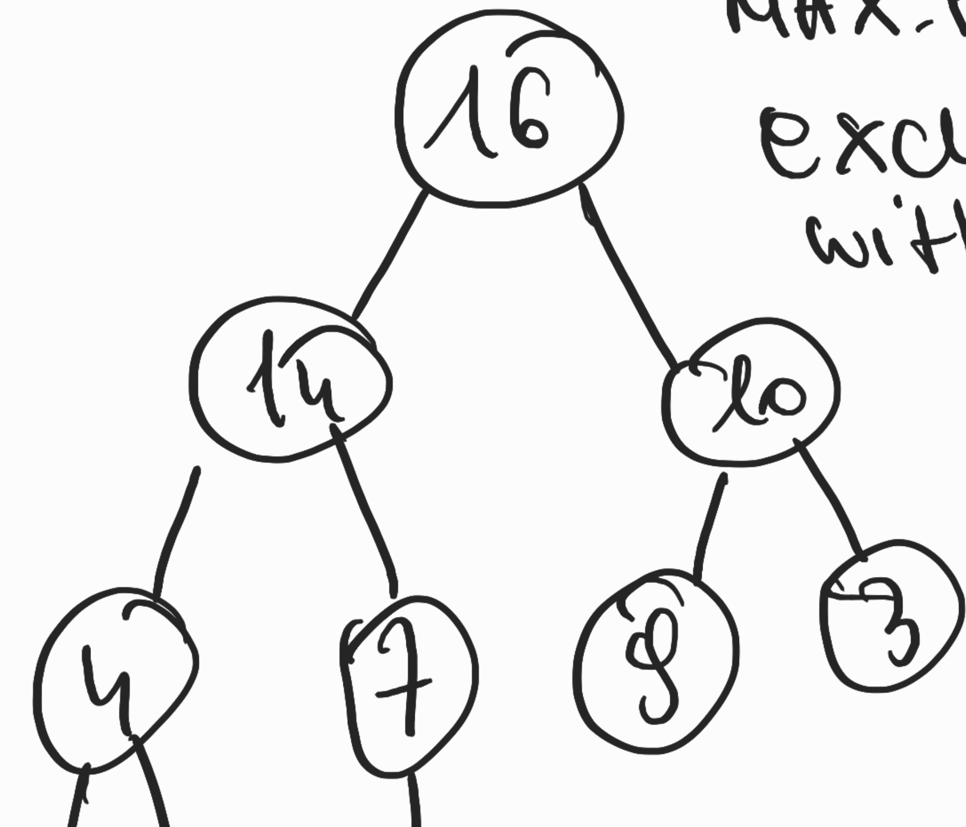
Assume that the trees rooted at left(i) and right(i) are max.heaps

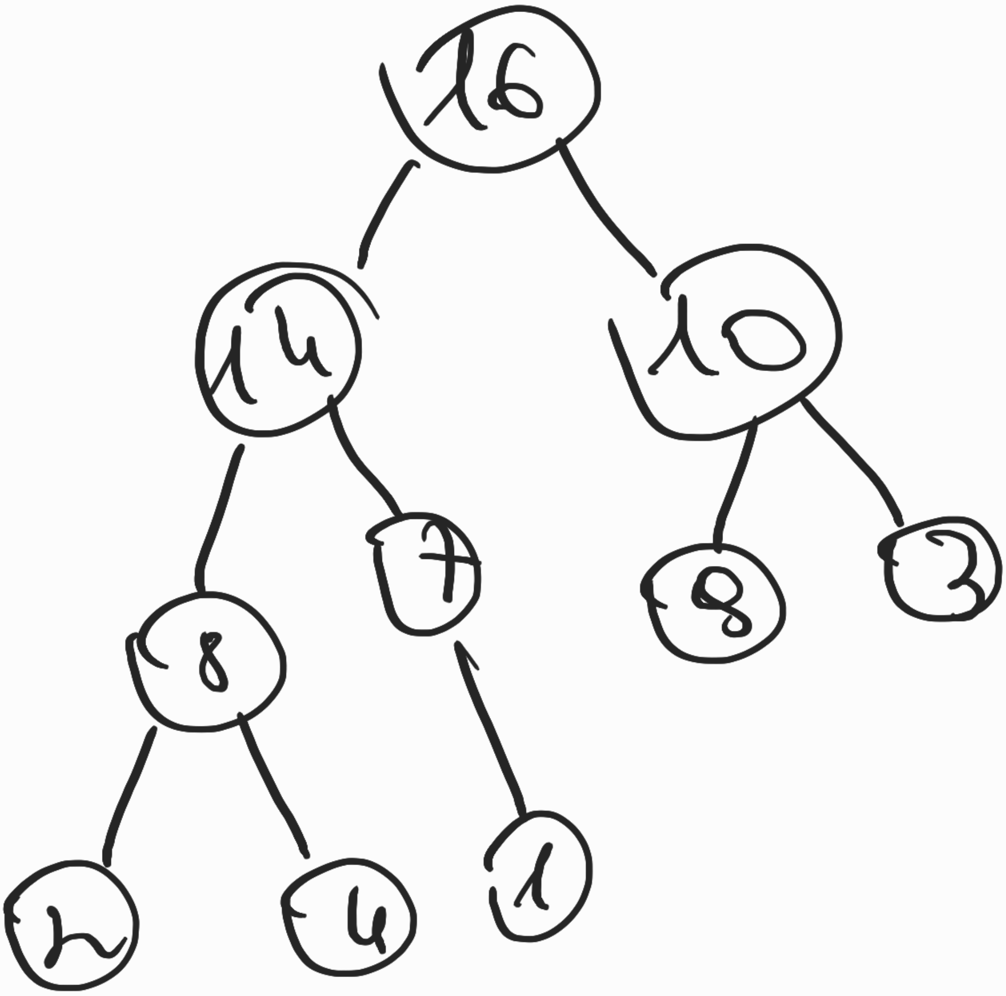
EXAMPLE:



MAX_HEPIFY (A,2)
heapsize=10
EXCHANGE A[2] with A[4]

MAX_HEAPIFY (A,4)
exchange A[4] with A[8]





Convert A[i...n] into a max-heap.

Build max_heap(A):
    for $i = n/2$ downto 1
        max_heapify(A,i)

↓ WHY?

$A[n/2+1 \ldots n]$
are leaves.

$O(n \lg n)$

OBSERVE Max-Heapify
takes $\Theta(1)$ for nodes
that are one level
above the leaves
and in general $\Theta(l)$
times for nodes that
are l levels above the
leaves

OBSERVE
$n/4$ nodes with level 1,
$n/8$ with level 2

$\vdots$

Total amount of work in the for loop

$n/4\,(1\,c) + n/8\,(2\,c) + n/16\,(3\,c) + \ldots + 1\,(\lg n\; c)$

Set $n/n = 2^k$

$$c \cdot 2^k \underbrace{\left(1/2^0 + \frac{2}{2^1} + \frac{3}{2^2} + \ldots \frac{(k+1)}{2^k}\right)}_{\text{bounded by constant}}$$

$$\downarrow$$

$$\sum_{i=0}^{k} \frac{i+1}{2^i}$$

1) Build_max_heap from unordered array $O(n)$

2) Find max_element $A[i]$ $O(1)$

3) Swap elements $A[n]$ with $A[i]$ $O(1)$

(max element is now at the end of array)

4) Discard node n from heap incrementing heap size

5) New root may violate max heap but children are max. heaps - maxheapify $O(\lg n)$

2 1 2 4 2 1

4, 2, 1