Heaps

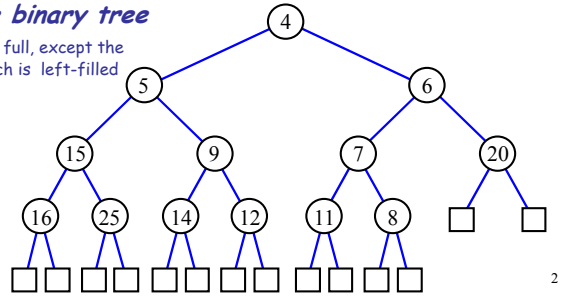
- Heaps
- Properties
- Deletion, Insertion, Construction
- Implementation of the Heap
- Implementation of Priority Queue using a Heap
- An application: HeapSort

Heaps (Min-heap)

Complete binary tree that stores a collection of keys (or key-element pairs) at its internal nodes and that satisfies the additional property:

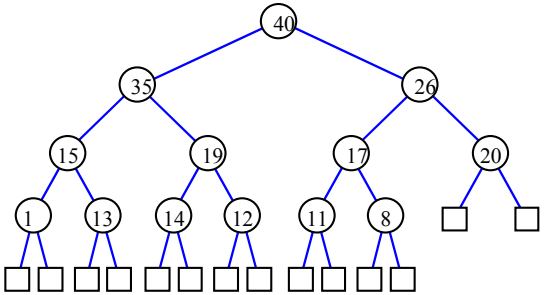
key(parent) ≤ key(child)

REMEMBER:
complete binary tree
all levels are full, except the last one, which is left-filled

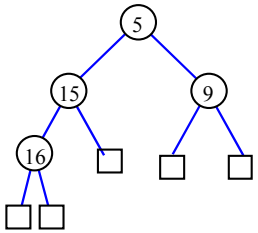


Max-heap

key(parent) ≥ key(child)



We store the keys in the internal nodes only

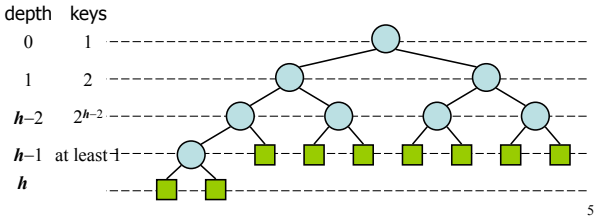


After adding the □ leaves the resulting tree is full

Height of a Heap

- Theorem: A heap storing n keys has height $O(\log n)$

  Proof:
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-2$ and at least one key at depth $h-1$, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-2} + 1$
  - Thus, $n \geq 2^{h-1}$, i.e., $h \leq \log n + 1$



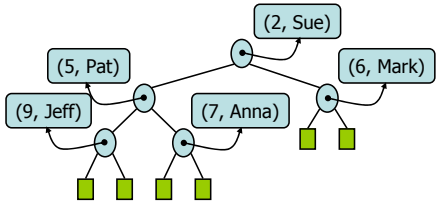
Notice that ….

- We could use a heap to implement a priority queue
- We store a (key, element) item at each internal node

  removeMin():

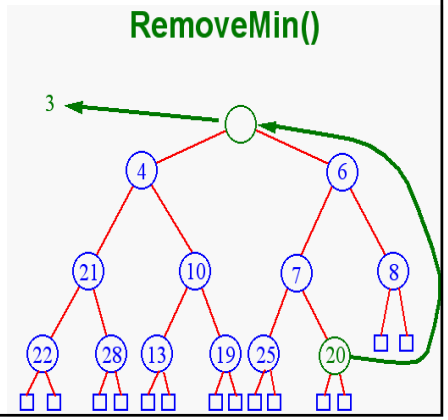
  → Remove the root

  → Re-arrange the heap!

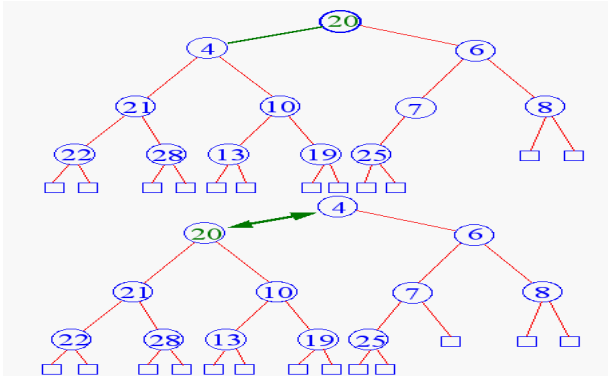


Removal From a Heap

- The removal of the top key leaves a hole
- We need to fix the heap
- First, replace the hole with the last key in the heap
- Then, begin Downheap …

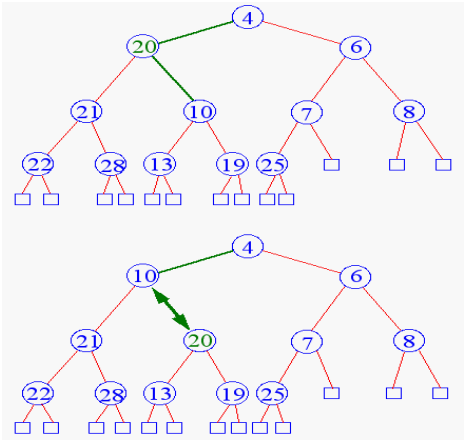


Downheap

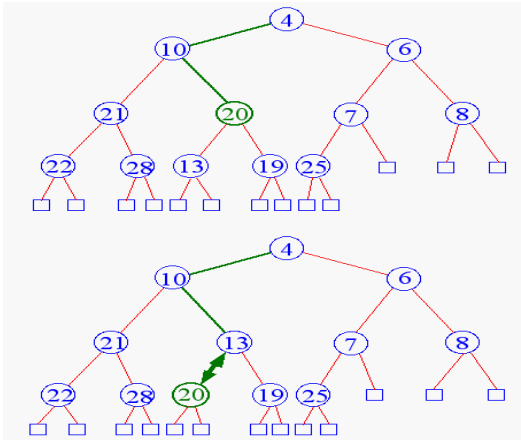


- Downheap compares the parent with the smallest child. If the child is smaller, it switches the two.

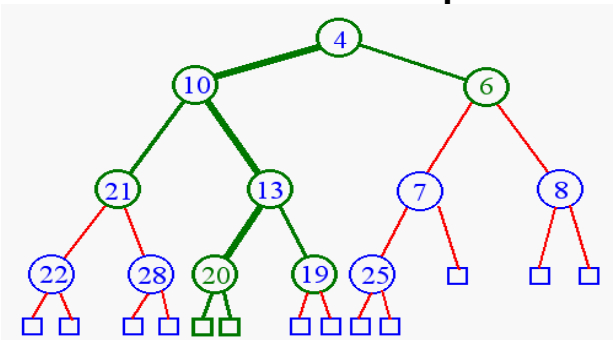
Downheap Continues



Downheap Continues



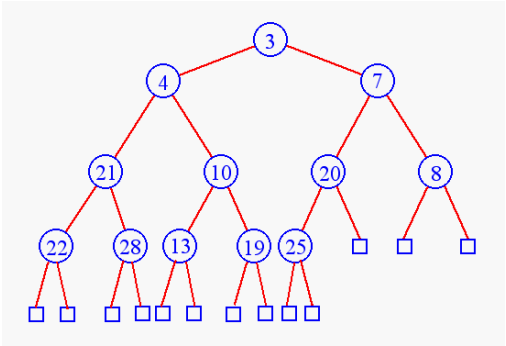
End of Downheap



- Downheap terminates when the key is greater than the keys of both its children or the bottom of the heap is reached.
  - (total #swaps) ≤ $(h - 1)$, which is O(log n)

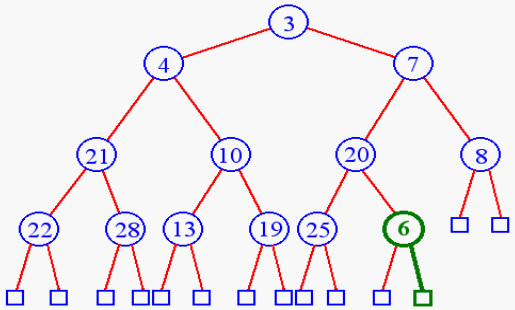
Heap Insertion

The key to insert is 6



Heap Insertion

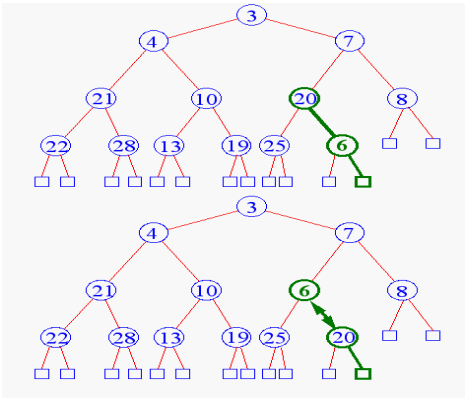
Add the key in the ***next available position*** in the heap.



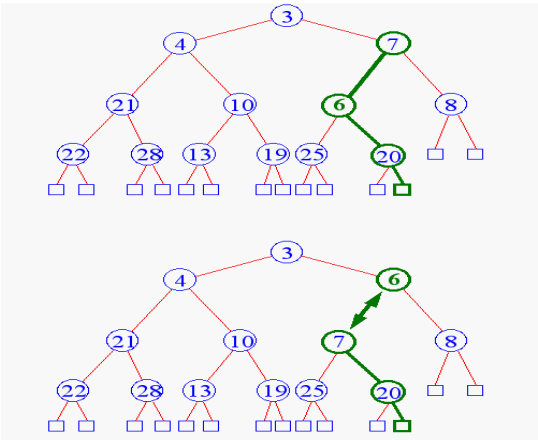
Now begin ***Upheap***.

Upheap

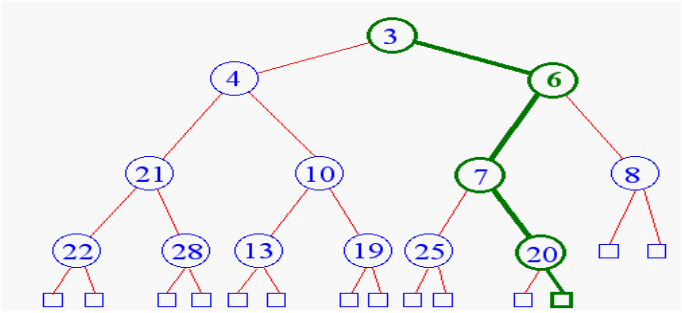
- Swap parent-child keys out of order



Upheap Continues



End of Upheap



- Upheap terminates when new key is greater than the key of its parent or the top of the heap is reached
- (total #swaps) $\leq$ (h - 1), which is O(log n)

Heap Construction

We could insert the Items one at the time with a sequence of Heap Insertions:

$$\sum_{k=1}^{n} \log k = O(n \log n)$$

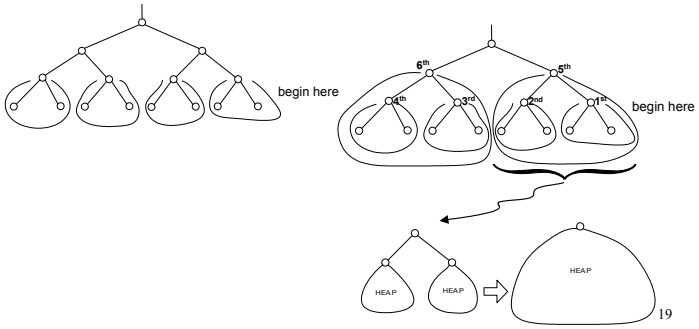
But we can do better ….

Bottom-up Heap Construction

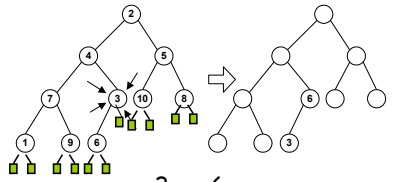
- We can construct a heap storing ***n*** given keys using a bottom-up construction

Construction of a Heap

Idea: Recursively re-arrange each sub-tree in the heap starting with the leaves



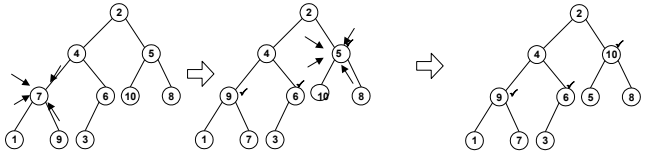
Example 1 (Max-Heap)



--- keys already in the tree ---

$\underline{3} \leftrightarrow 6$

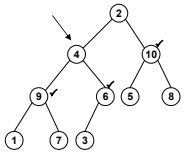
I am not drawing the leaves anymore here



$\underline{7} \leftrightarrow 9$

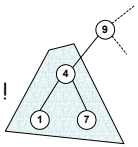
$\underline{5} \leftrightarrow 10$

Example 1

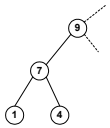


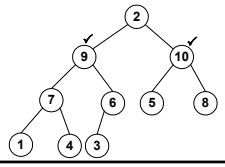
$4 \leftrightarrow 9$

This is not a heap !

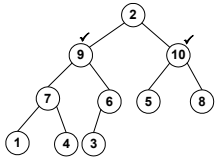


$4 \leftrightarrow 7$

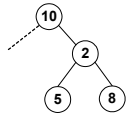


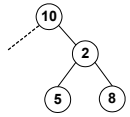


Example 1

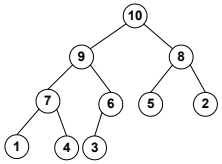


Finally: $2 \leftrightarrow 10$





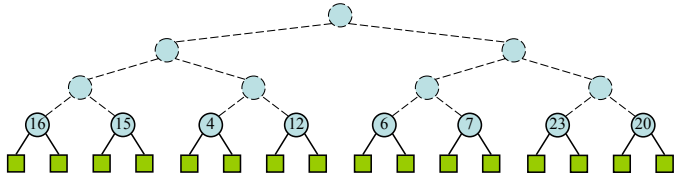
$2 \leftrightarrow 8$

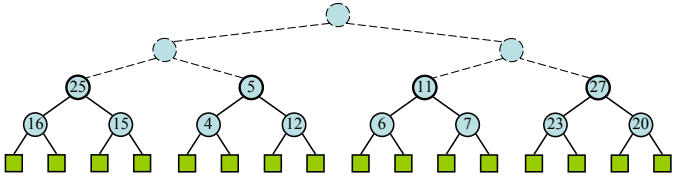


--- keys given one at a time ---

Example 2 (min-heap)

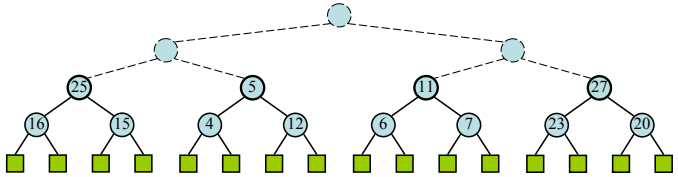
[20,23,7,6,12,4,15,16,27,11,5,25,8,7,10]

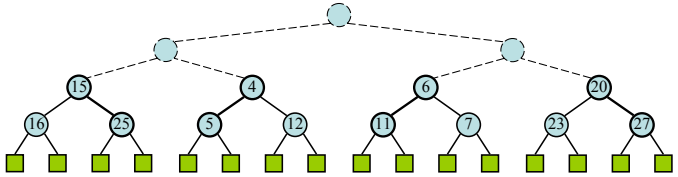




Example 2

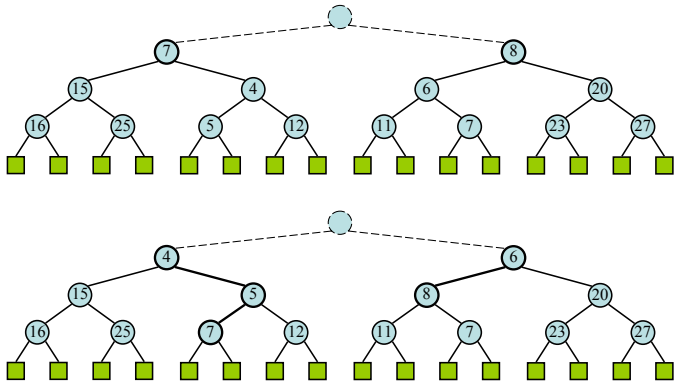
20,23,7,6,12,4,15,16,27,11,5,25,8,7,10]





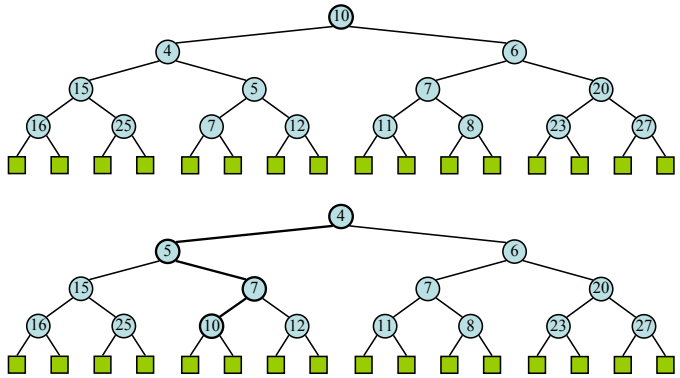
Example 2

20,23,7,6,12,4,15,16,27,11,5,25,8,7,10]



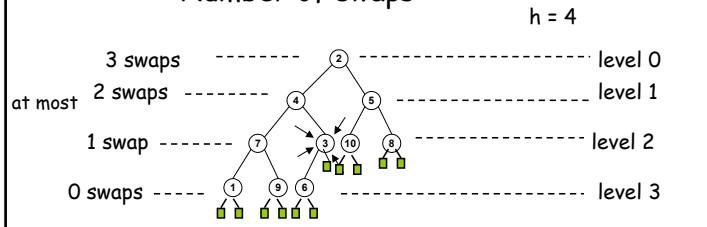
Example 2

20,23,7,6,12,4,15,16,27,11,5,25,8,7,10]



Analysis of Heap Construction

Number of swaps

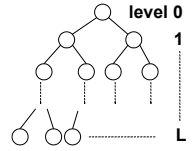


Let L be the max level

level i -------- L - i swaps

Analysis of Heap Construction

Number of swaps?



At level i, the number of swaps is

$\leq$ L - i for each node

At level i, there are $\leq 2^i$ nodes

Total: $\leq \sum_{i=0}^{L} (L - i) \cdot 2^i$

Calculating $O(\Sigma(L - i)\cdot 2^i)$

Let $j = L-i$, then $i = L-j$ and

$$\sum_{i=0}^{L}(L - i)\cdot 2^i = \sum_{j=0}^{L} j\, 2^{L-j} = 2^L \sum_{j=0}^{L} j\, 2^{-j}$$

Consider $\Sigma j\cdot 2^{-j}$:

$\Sigma\, j\cdot 2^{-j} = 1/2 + 2\ 1/4 + 3\ 1/8 + 4\ 1/16 + \cdots$

$= 1/2 + 1/4 + 1/8 + 1/16 + \cdots <= 1$

$+ \quad 1/4 + 1/8 + 1/16 + \cdots <= 1/2$

$+ \quad\quad + 1/8 + 1/16 + \cdots <= 1/4$

$\Sigma j\cdot 2^{-j} <= 2$

So $2^L \Sigma j\, 2^{-j} <= 2\cdot 2^L = 2^{L+1}$

$$2^L \sum_{j=1}^{L} j/2^j \leq 2^{L+1}$$

So, the number of swaps is $\leq O(n)$

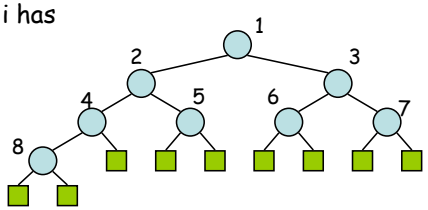
Implementing a Heap with an Array

A heap can be nicely represented by a vector (array-based), where the node at rank i has

- left child at rank 2i

and

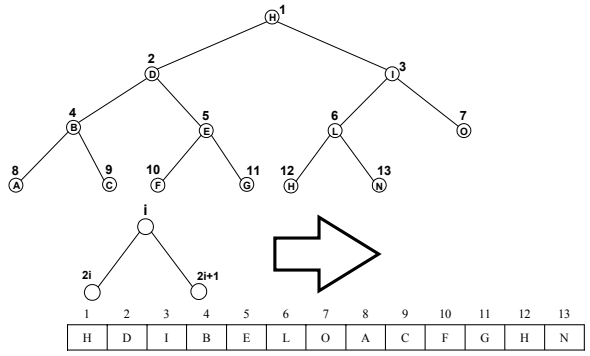
- right child at rank 2i + 1



1	2	3	4	5	6	7	8

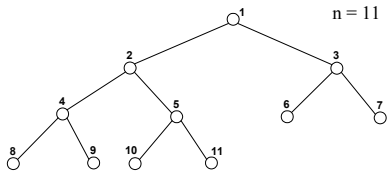
The leaves do no need to be explicitly stored

Example

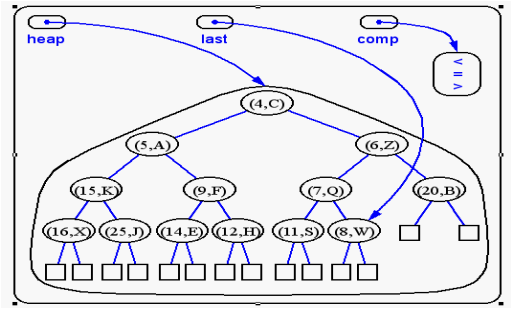


Reminder .....

Left child of T[i]	T[2i]	if	$2i \le n$
Right child of T[i]	T[2i+1]	if	$2i + 1 \le n$
Parent of T[i]	T[i div 2]	if	$i > 1$
The Root	T[1]	if	$T \ne 0$
Leaf? T[i]	TRUE	if	$2i > n$



Implementation of a Priority Queue with a Heap



(up-heap bubbling)

insert	O(log n)
min	O(1)
removeMin	O(log n)

(remove root + down-heap bubbling)

Each PQ ADT method can be performed in O(1) or O(log n) time.

Application: Sorting Heap Sort

Construct initial heap $O(n)$

n times

remove root	$O(1)$
re-arrange	$O(\log n)$
remove root	$O(1)$
re-arrange	$O(\log (n-1))$
...	⋮
...	

When there are i nodes left in the PQ: $\lfloor \log i \rfloor$

$\rightarrow$TOT = $\sum_{i=1}^{n} \lfloor \log i \rfloor$

= O(n log n)