Lecture 7-2: Heap

Sunghyun Cho
Professor
Division of Computer Science
chopro@hanyang.ac.kr

HANYANG UNIVERSITY

Recall Priority Queue ADT

- A priority queue stores a collection of entries
- Each **entry** is a pair (key, value)
- Main methods of the Priority Queue ADT
  - insert(k, x)
    inserts an entry with key k and value x
  - removeMin()
    removes and returns the entry with smallest key
- Additional methods
  - min()
    returns, but does not remove, an entry with smallest key
  - size(), isEmpty()
- Applications:
  - Standby flyers
  - Auctions
  - Stock market

Recall PQ Sorting

- We use a priority queue
  - Insert the elements with a series of insert operations
  - Remove the elements in sorted order with a series of removeMin operations
- The running time depends on the priority queue implementation:
  - Unsorted sequence gives selection-sort: **O(n^2)** time
  - Sorted sequence gives insertion-sort: **O(n^2)** time
- Can we do better?

```
Algorithm PQ-Sort(S, C)
    Input sequence S, comparator C
    for the elements of S
    Output sequence S sorted in
    increasing order according to C
    P ← priority queue with
        comparator C
    while ¬S.isEmpty ()
        e ← S.remove (S. first ())
        P.insertItem(e, e)
    while ¬P.isEmpty()
        e ← P.removeMin().getKey()
        S.addLast(e)
```


Problems of Selection / Insertion Sort

- The Phases
  - Phase 1: build a queue from a input set
  - Phase 2: build a priority queue
- Problems
  - Selection sort: fast Phase 1 but very slow Phase 2
  - Insertion sort: very slow Phase 1 but fast Phase 2
- How can we balance the running times of the two phases?
  - An efficient realization of a priority queue uses a data structure called a heap.

Heap Data Structures

- Heap : A binary tree *T* satisfies two additional properties – **relational property** and **structural property**

(1) **Heap-order Property** : in a heap *T*, for every node *v* other than the root, the key stored at v is greater than or equal to the key stored at *v*'s parent

(2) **Complete binary tree property**: in a heap T with height h

- levels of 0,1,2,…,h-1 of T have the maximum number of nodes possible (level i has 2^i nodes , for $0 \le i \le h-1$)
- in the level h, all the internal nodes are to the left of the external node and there at most one node with one child, which must be a left child.

Heap Data Structures (cont.)

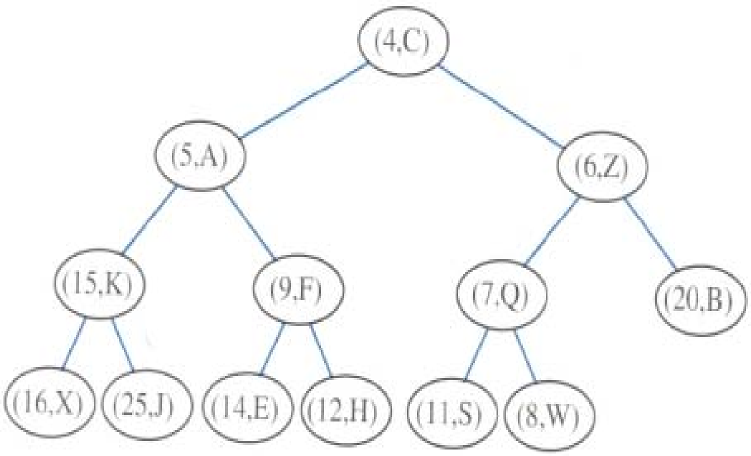
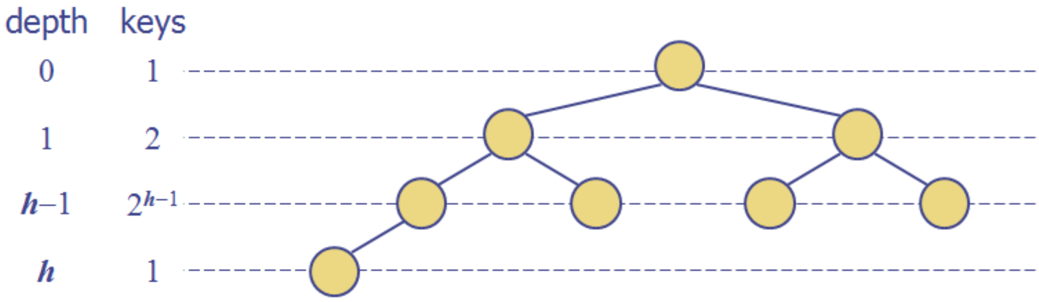


Figure 7.3: Example of a heap storing 13 entries with integer keys. The last node is the one storing entry (8, *W*).

- The last node of a heap is the rightmost node of maximum depth

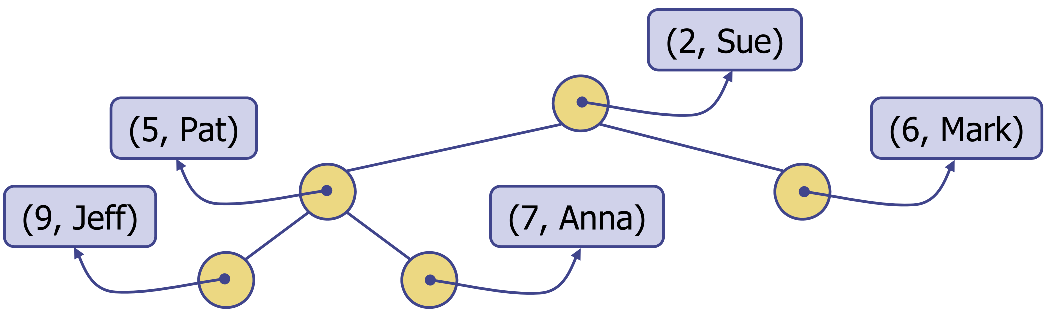
Height of a Heap

- **Theorem**: A heap storing n keys has height $O(\log n)$
  Proof: (we apply the complete binary tree property)
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$



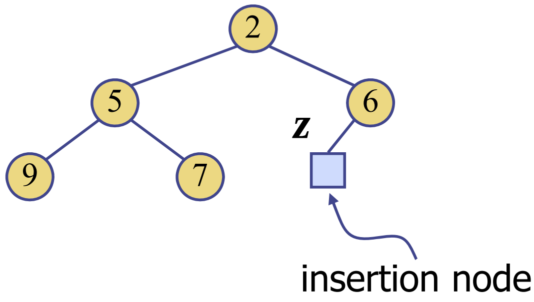
Heaps and Priority Queues

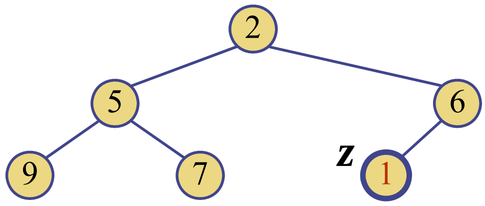
- We can use a heap to implement a priority queue
- We store a (key, element) item at each internal node
- <u>**We keep track of the position of the last node (definitely including the position of the root)**</u>



Insertion into a Heap

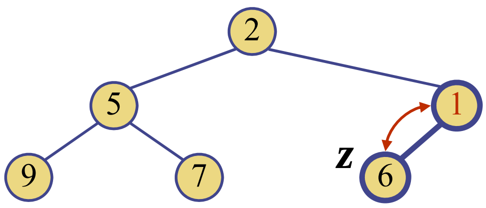
- Method insertItem of the priority queue ADT corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
  - Store k at z
  - Restore the heap-order property (discussed next)

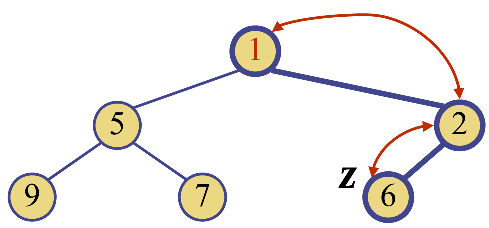




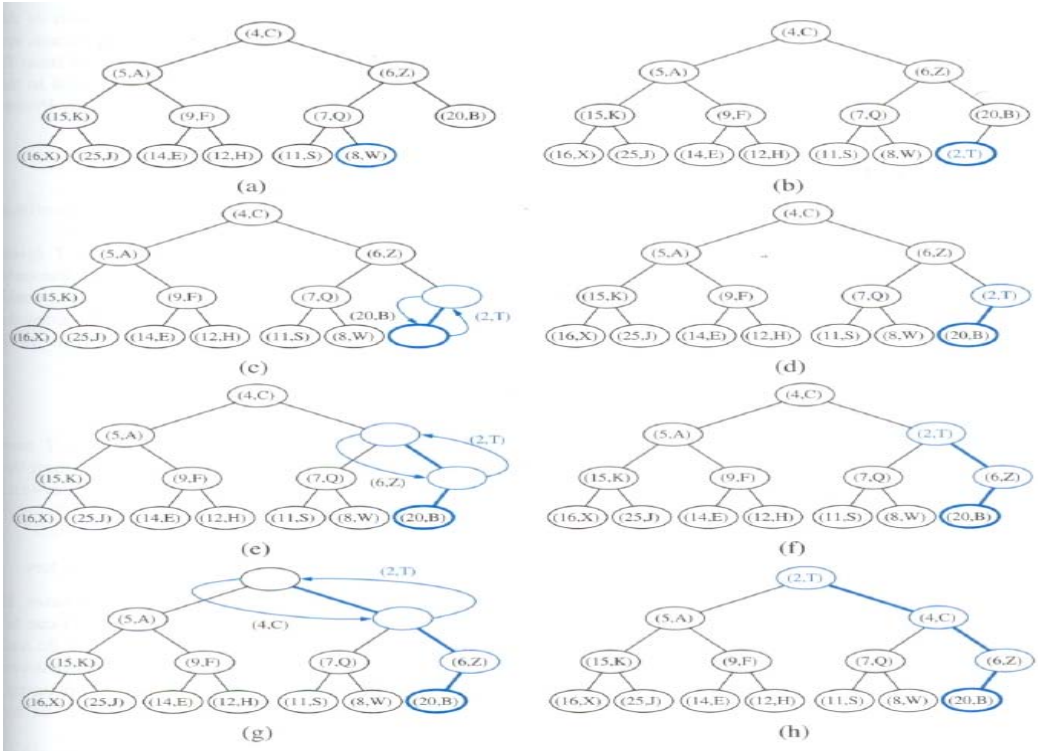
Upheap

- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time



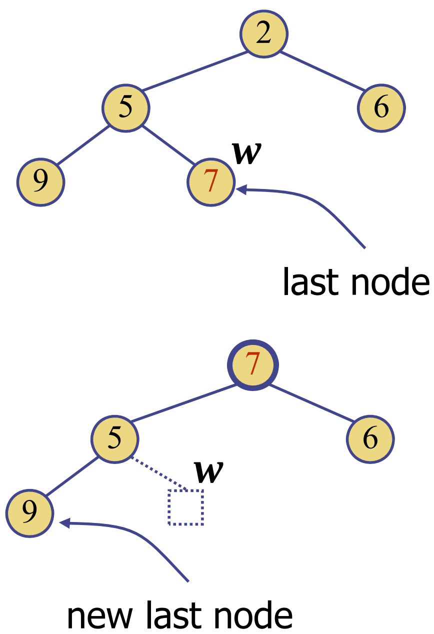


Example of Insertion



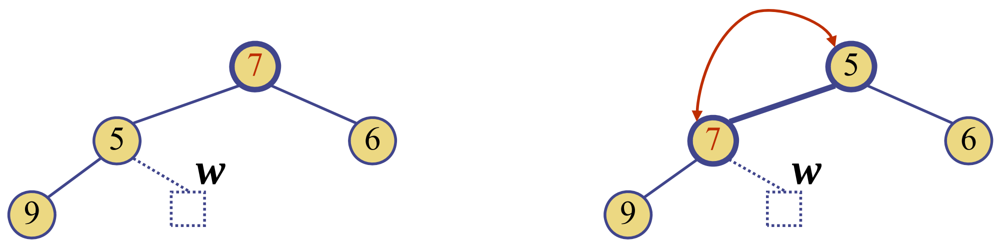
Removal from a Heap

- Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node ***w***
  - Remove ***w***
  - Restore the heap-order property (discussed next)

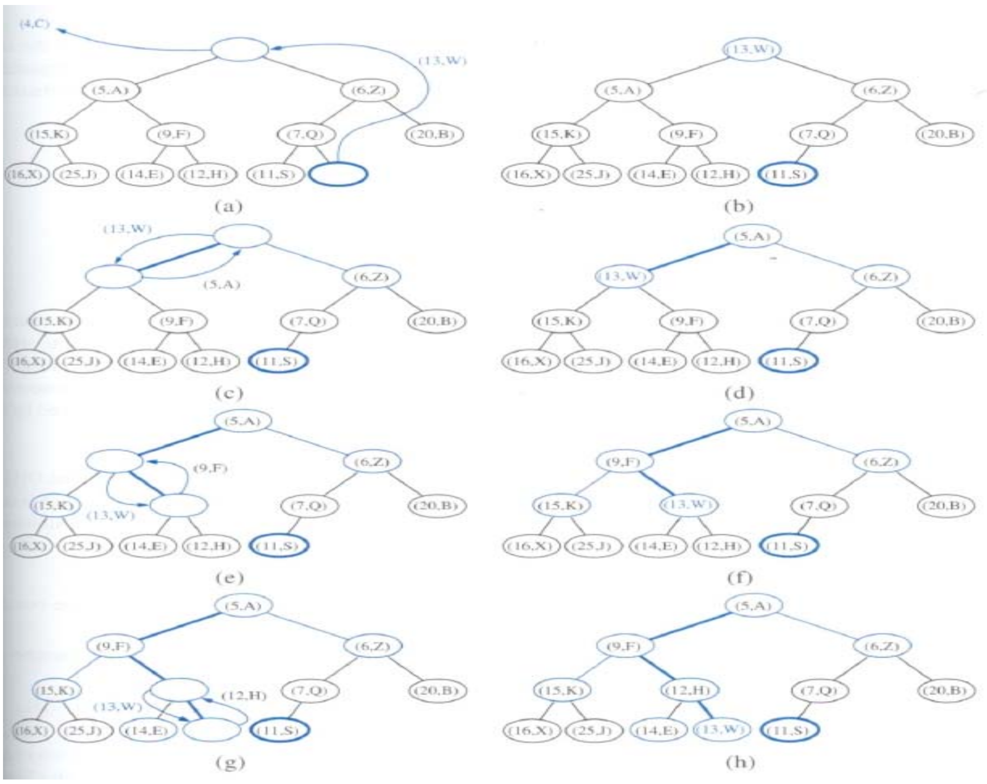


Downheap

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Downheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time

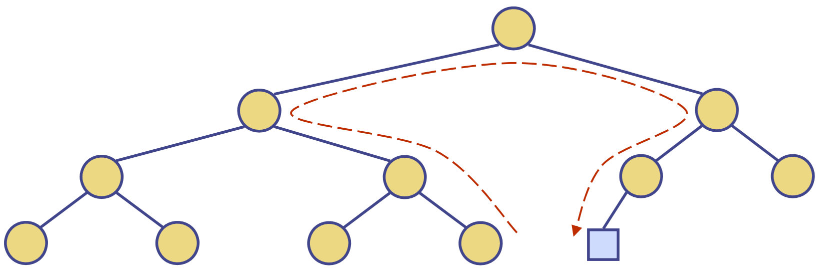


Example of Removal



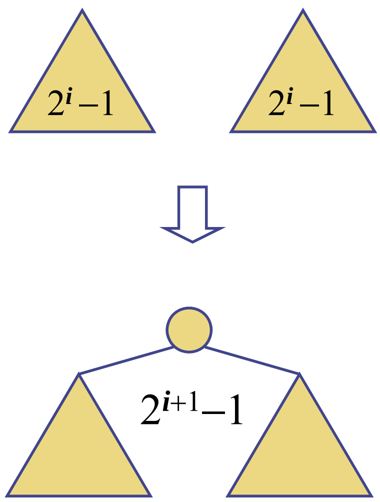
Updating the Last Node

- The insertion node can be found by traversing a path of $O(\log n)$ nodes
  - Go up until a left child or the root is reached
  - If a left child is reached, go to the right child
  - Go down left until a leaf is reached
- Similar algorithm for updating the last node after a removal

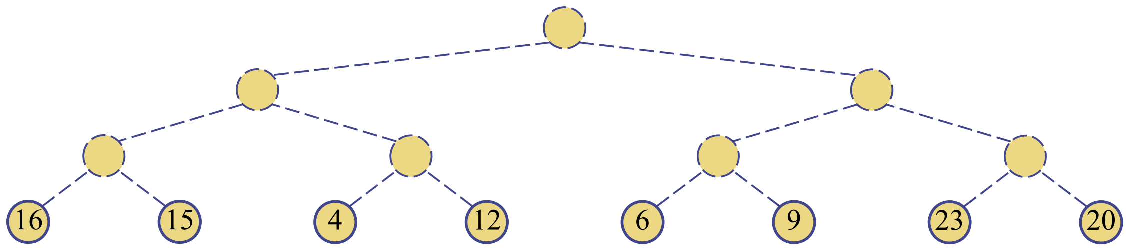


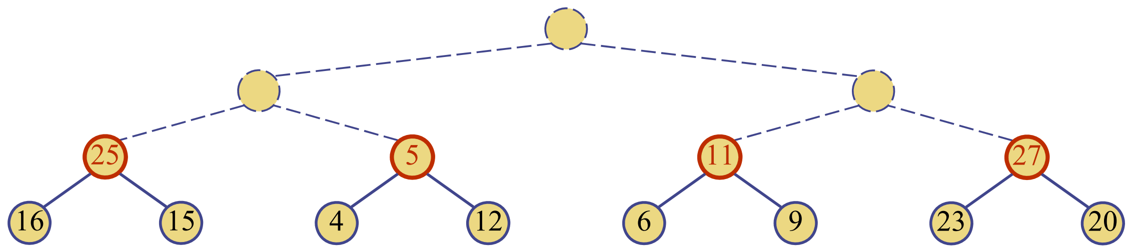
Bottom-up Heap Construction

- We can construct a heap storing n given keys in using a bottom-up construction with $\log n$ phases
- In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys

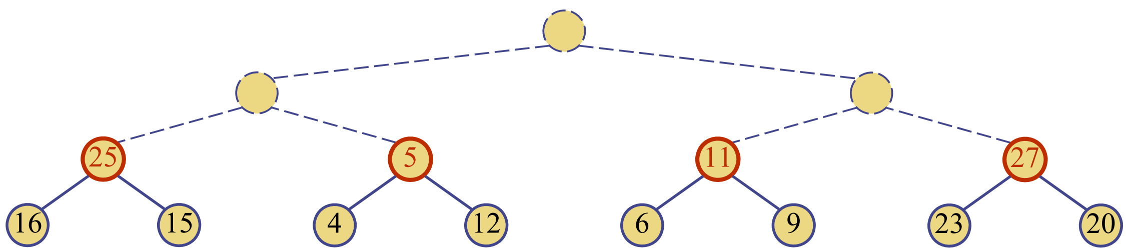


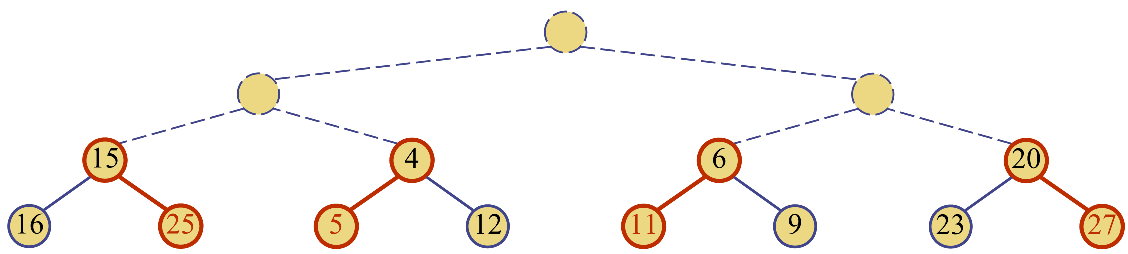
Example



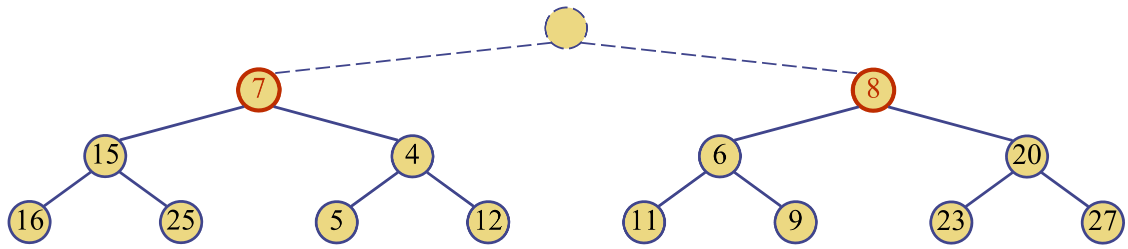


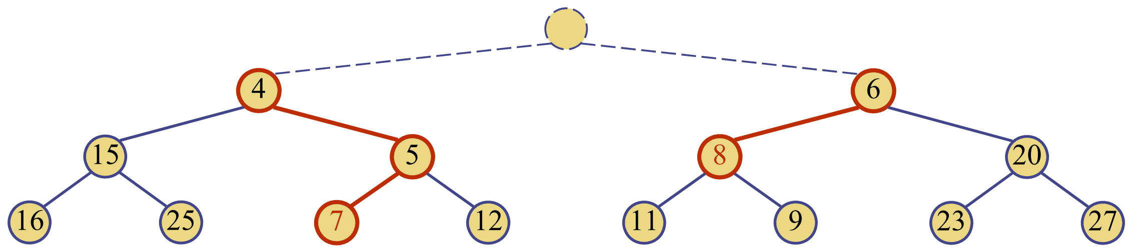
Example (contd.)



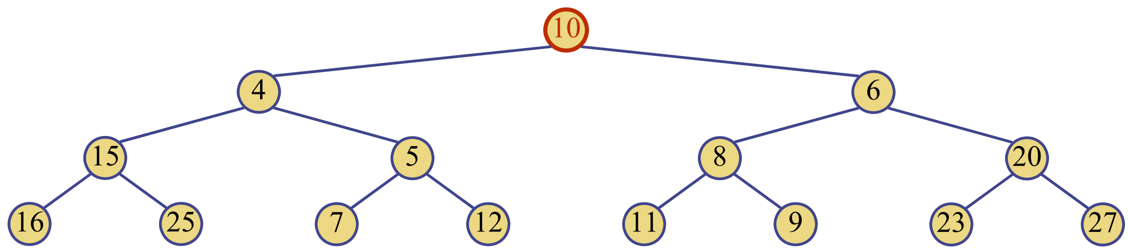


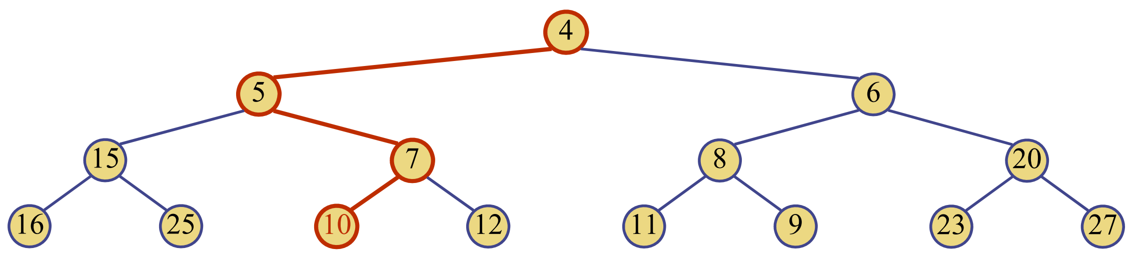
Example (contd.)





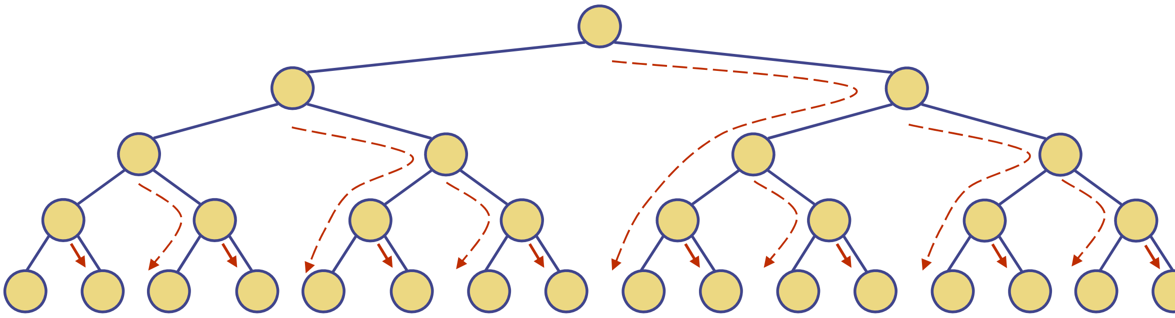
Example (end)





Analysis

- We visualize the worst-case time of a downheap with a proxy path that goes first right and then repeatedly goes left until the bottom of the heap (this path may differ from the actual downheap path)
- Since each node is traversed by at most two proxy paths, the total number of nodes of the proxy paths is $O(n)$
- Thus, bottom-up heap construction runs in $O(n)$ time
- Bottom-up heap construction is faster than n successive insertions and speeds up the first phase of heap-sort



Heap-Sort

- Consider a priority queue with n items implemented by means of a heap
  - the space used is $O(n)$
  - methods insert and removeMin take $O(\log n)$ time
  - methods size, isEmpty, and min take time $O(1)$ time
- Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time
- The resulting algorithm is called heap-sort
- Heap-sort is much faster than quadratic sorting algorithms, such as insertion-sort and selection-sort

In-Place Heap Sort

- Assume the sequence is implemented by means of an array.
  - Speed up heap-sort
  - Reduce its space requirement by a constant factor
    - ➢ Rearrange elements of the sequence instead of transferring them out of the sequence and then back in.
    - ➢ **<u>In-Place implies that only additional O(1) extra DS items are required to solve the problem</u>**

- In-Place Heap-Sort Algorithm
  - At any time during the execution of the algorithm, we use the left portion of S, up to a certain rank i-1, to store the entries of the heap, and the right portion of S, from rank i to n, to store the unsorted entries of the sequence. Thus, the first i elements of S (at rank 1,….,i) provide the vector representation of the heap.
  - Start with an empty heap and move the boundary between the heap and the sequence from left to right, one step at a time. In step I (i=1,….n), we expand the heap by adding the element at the rank i.

In-Place Heap Sort (cont.)

	4	7	2	1	5	3
	4	7	2	1	5	3
	2	4	7	1	5	3
	1	2	4	7	5	3
	1	2	4	5	7	3
	1	2	3	4	5	7

Merging Two Heaps

- We are given two heaps and a key ***k***
- We create a new heap with the root node storing ***k*** and with the two heaps as subtrees
- We perform downheap to restore the heap-order property