بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۲۵

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

Heaps

Recall Priority Queue ADT

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

Recall PQ Sorting

IUT-ECE

- 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? Balancing the above

Algorithm PriorityQueueSort(L, P):

Input: An STL list L of n elements and a priority queue, P, that compares elements using a total order relation

Output: The sorted list L

while !L.empty() **do**
    $e \leftarrow L$.front
    L.pop_front() {remove an element e from the list}
    P.insert(e) {... and it to the priority queue}
while !P.empty() **do**
    $e \leftarrow P$.min()
    P.removeMin() {remove the smallest element e from the queue}
    L.push_back(e) {... and append it to the back of L}

We will have these results soon …

List-based

Operation	*Unsorted List*	*Sorted List*
size, empty	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$
min, removeMin	$O(n)$	$O(1)$

Heap-based

Operation	*Time*
size, empty	$O(1)$
min	$O(1)$
insert	$O(\log n)$
removeMin	$O(\log n)$

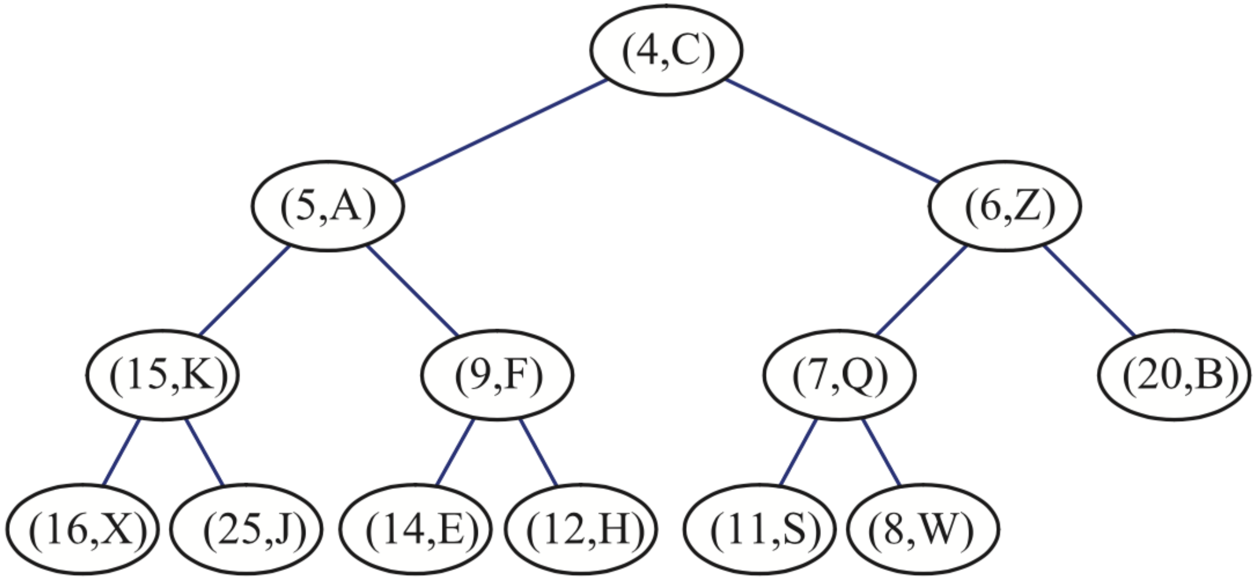
Heap: Overview

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:

1. Heap-order property

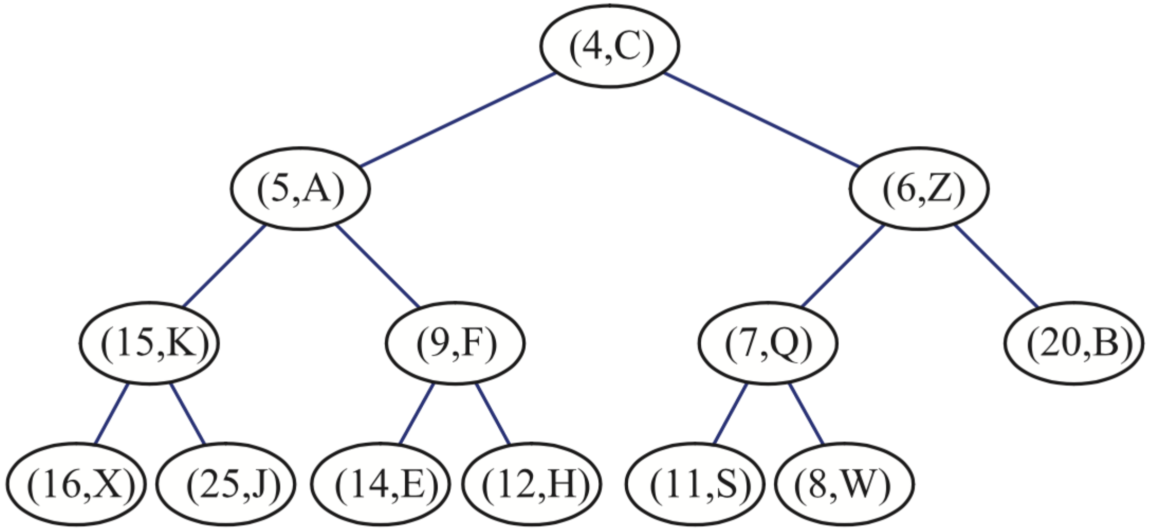
IUT-ECE

- 1. Heap-Order: for every internal node v other than the root,
  key(***v***) $\geq$ ***key***(***parent***(***v***))
  - The keys encountered on a path from the root to a leaf T are nondecreasing
  - A minimum key: always at the root



2. Complete binary tree property

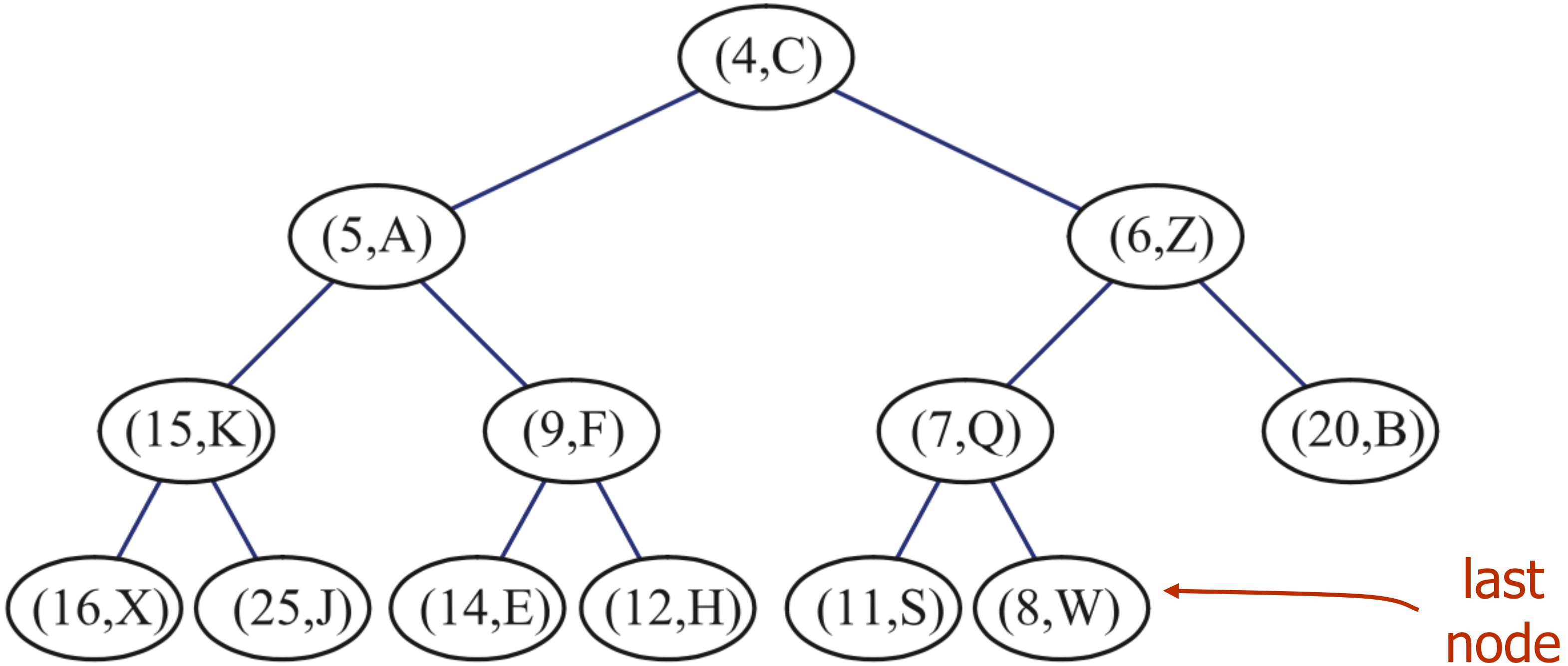
- Complete Binary Tree
  - Roughly speaking, every level, except for the last level, is completely filled, and all nodes in the last level are as far left as possible.
- let $\boldsymbol{h}$ be the height of the heap
  - for $\boldsymbol{i} = 0, \ldots, \boldsymbol{h} - 1$, there are 2^i nodes of depth $\boldsymbol{i}$
  - at depth $\boldsymbol{h} - 1$, the internal nodes are to the left of the external nodes



Heap: Overview

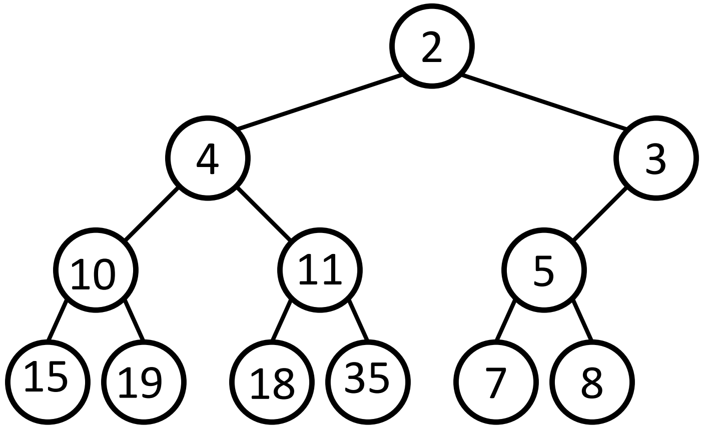
IUT-ECE

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
  - 1. Heap-order property
  - 2. Complete binary tree property
- The last node of a heap is the rightmost node of maximum depth



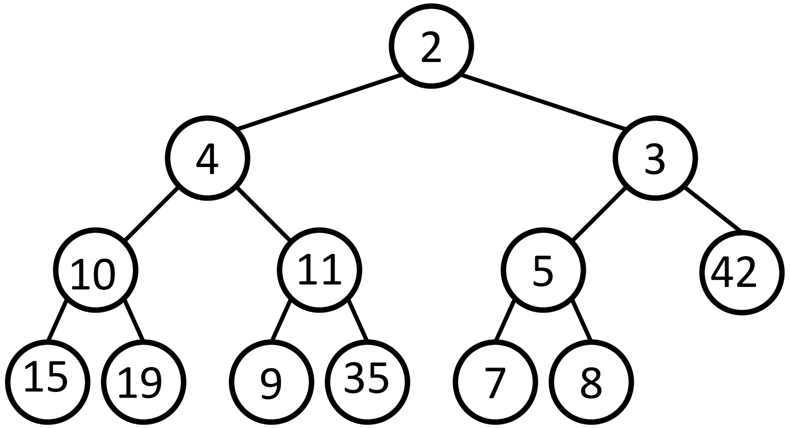
Example

- Min-Heap?



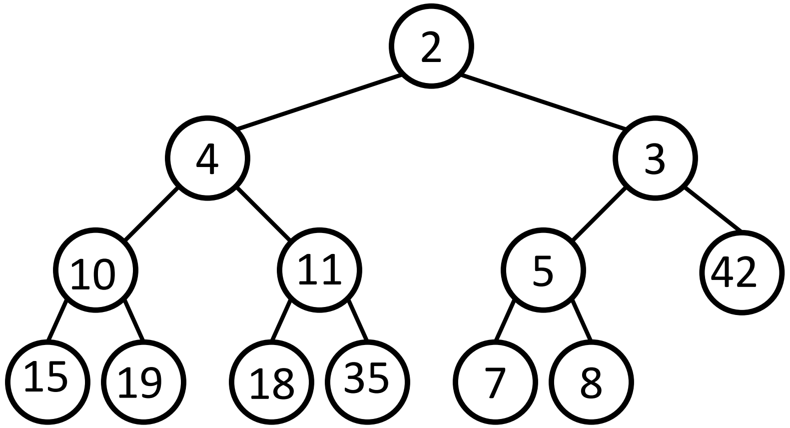
Example

- Min-Heap?



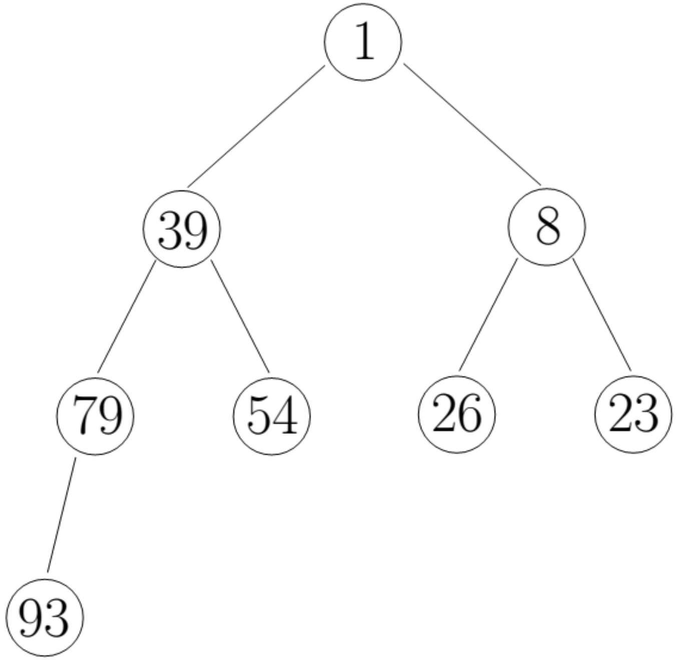
Example

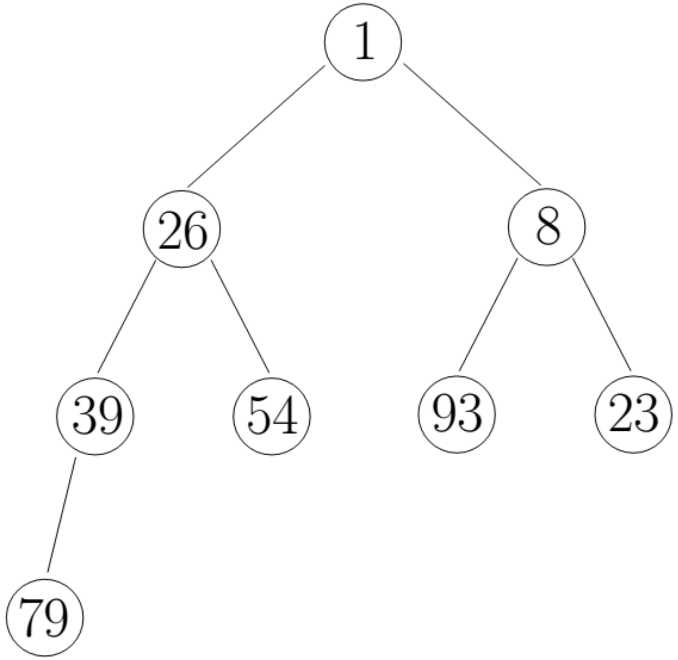
- Min-Heap?



Example

$$S = \{93, 39, 1, 26, 8, 23, \ 79, 54\}$$



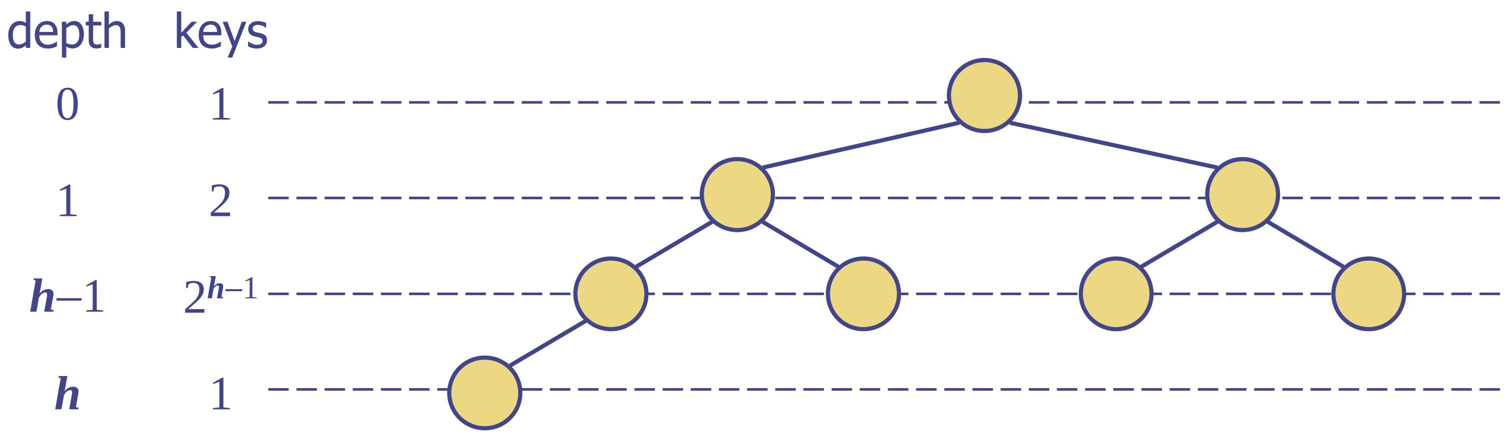


Height of a Heap of *n* elements

IUT-ECE

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

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

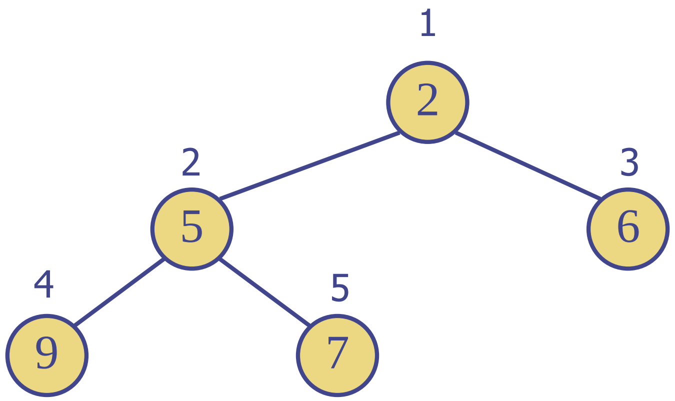


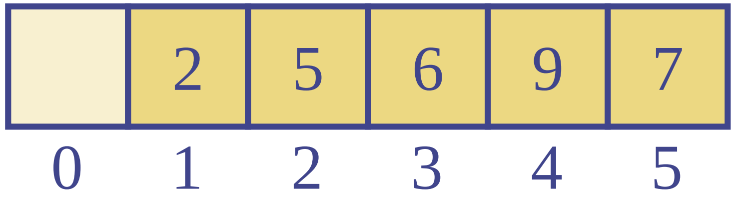
Heap

- دیدیم که heap به صورت درخت باینری تقریبا کامل است:
- پیاده سازی ساده با آرایه
- حفظ اوردر لگاریتمی

Vector-based Heap Implementation

- We can represent a heap with $\boldsymbol{n}$ keys by means of a vector of length $\boldsymbol{n} + 1$
- For the node at rank $\boldsymbol{i}$
  - the left child is at rank $2\boldsymbol{i}$
  - the right child is at rank $2\boldsymbol{i} + 1$
- Links between nodes are not explicitly stored
- The cell of at rank 0 is not used
- Last node at rank n





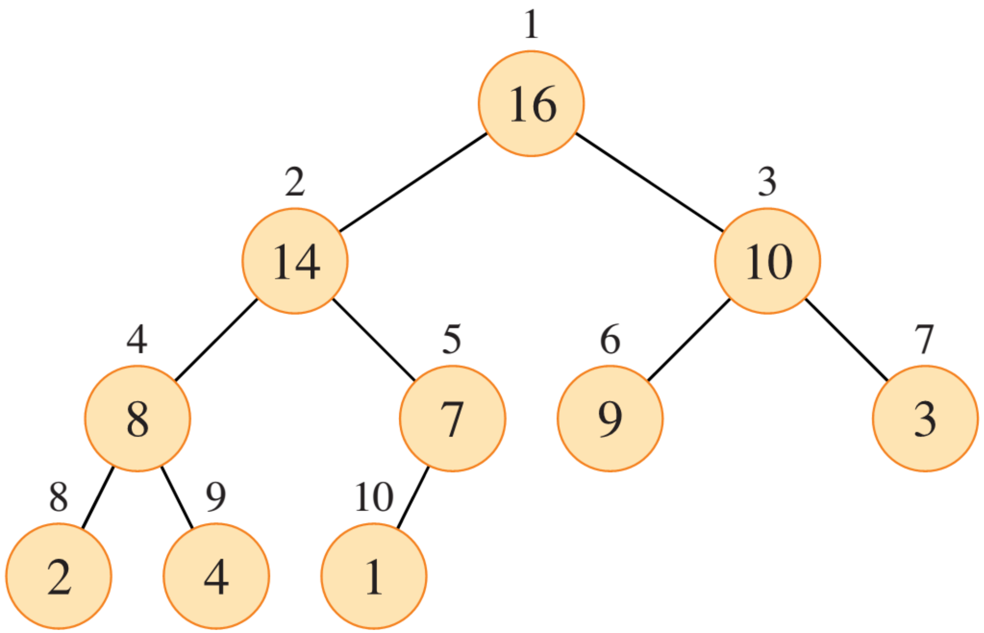
Example

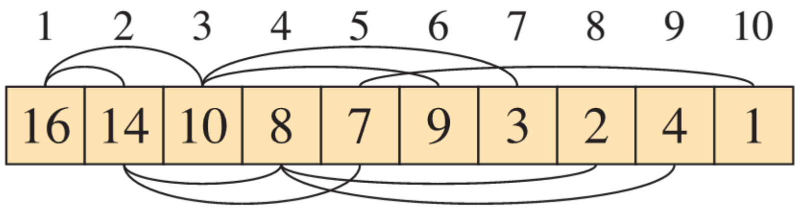
- Min heap?

[1,8,5,9,12,11,7]

Example

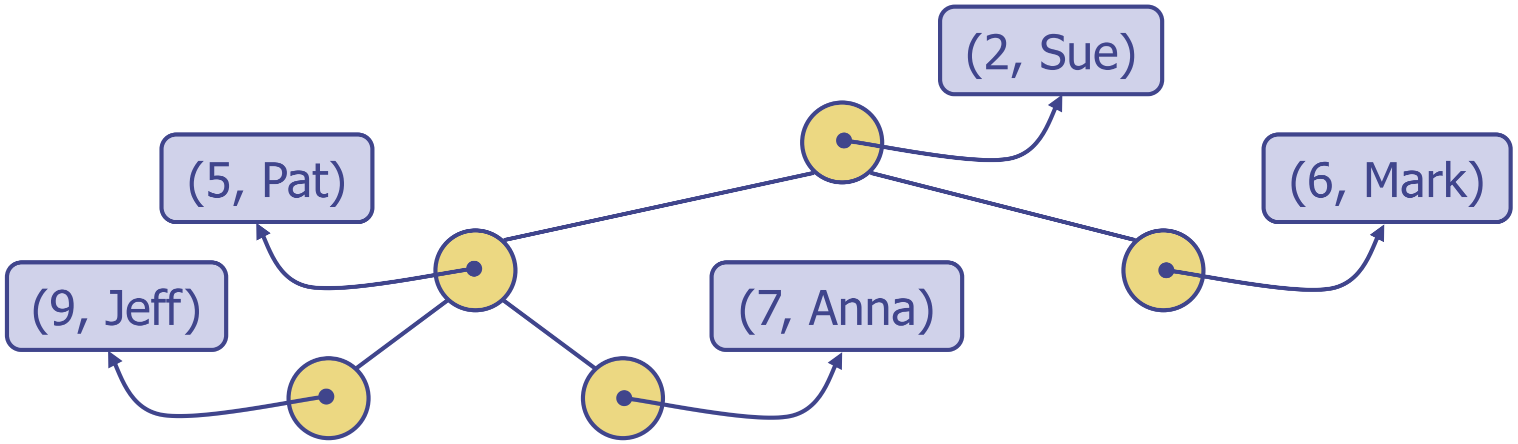
- Max heap?





Heaps and Priority Queues

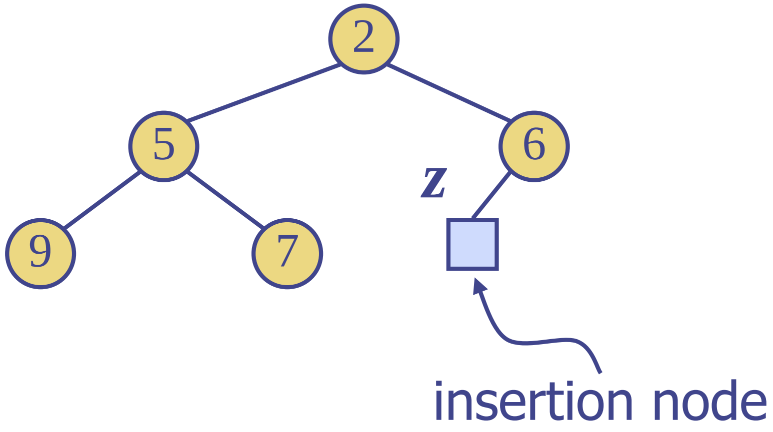
- We can use a heap to implement a priority queue
  - We say “heap-based PQ implementation”
- We store a (key, element) item at each internal node
- We keep track of the position of the last node
  - I am able to know who is the last node in O(1) time
  - Easy

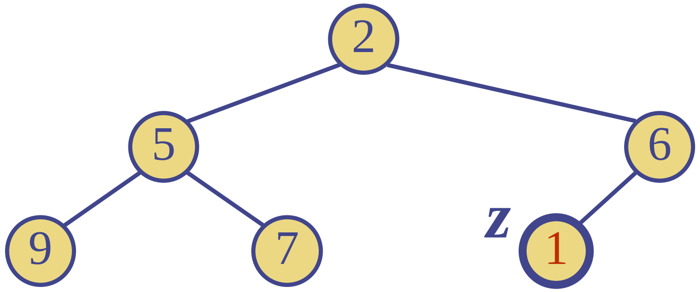


Insertion into a Heap

IUT-ECE

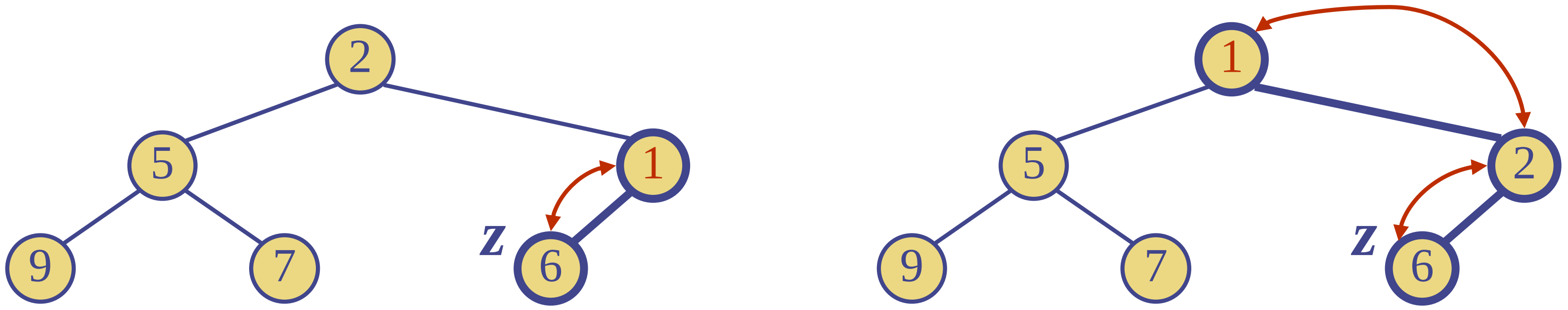
- Method **insert** of the priority queue ADT corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
    - How? discussed later
  - Store k at z
  - Restore the heap-order property (discussed next)





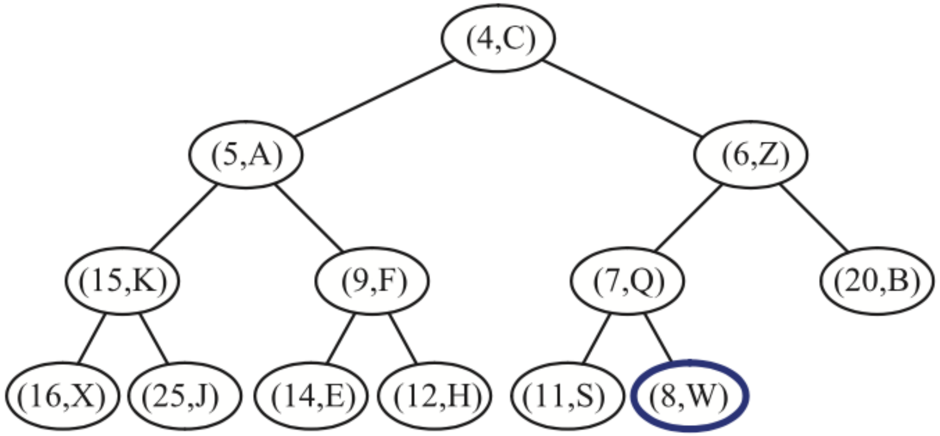
Upheap

- 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

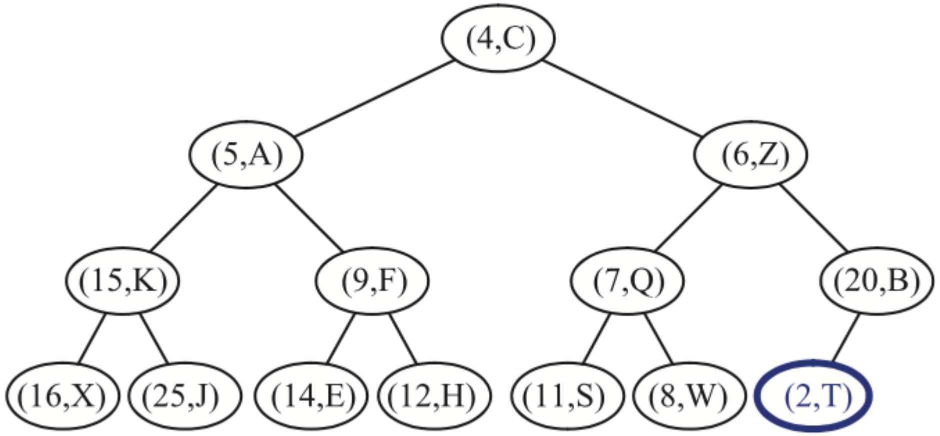


Insert: (2,T)

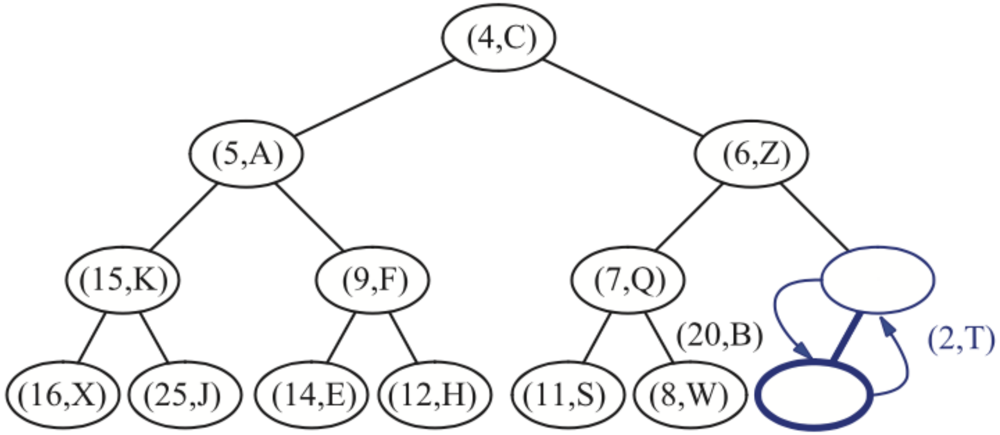
IUT-ECE



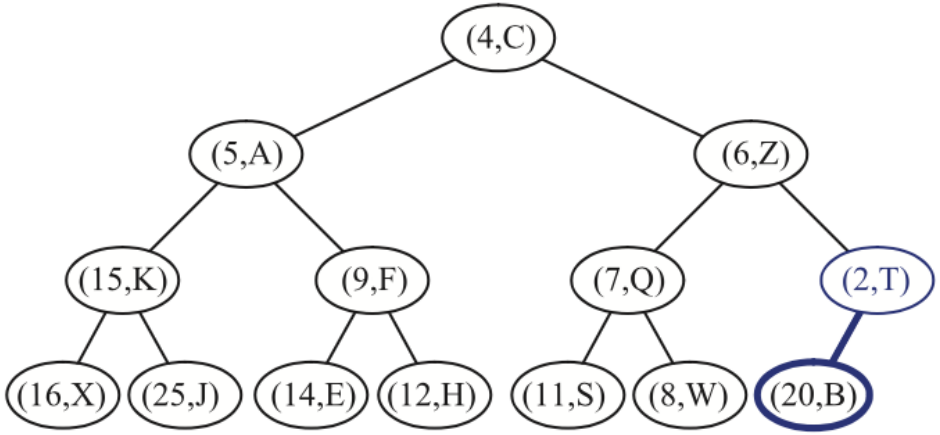
(a)



(b)



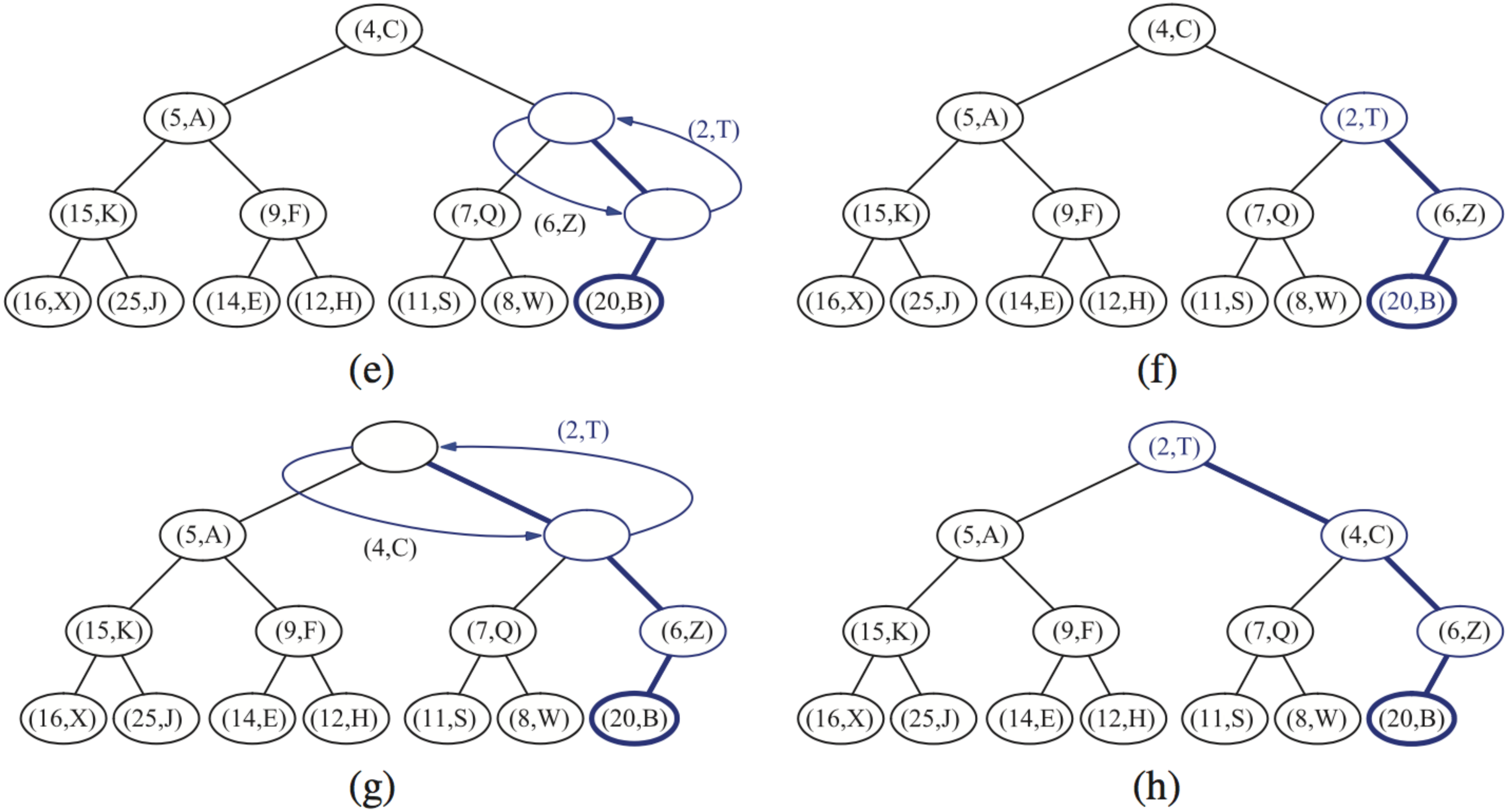
(c)



(d)

Insert: (2,T)

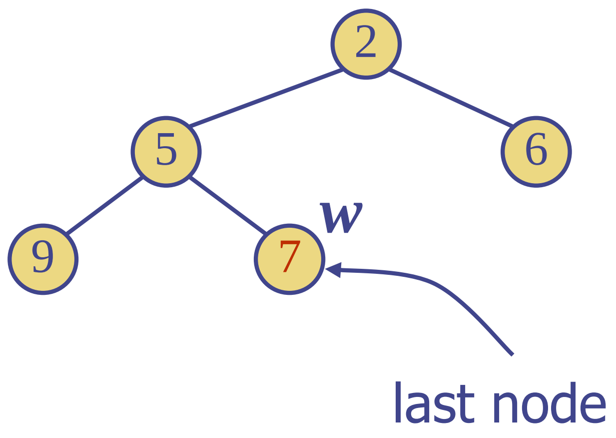
IUT-ECE

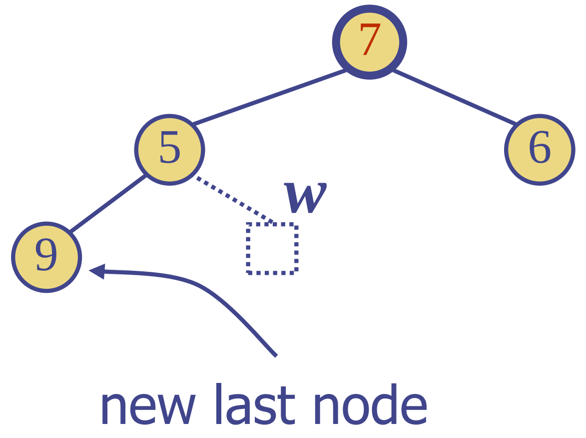


Removal from a Heap

- Method **removeMin** of the priority queue ADT corresponds to the removal of the root key from the heap

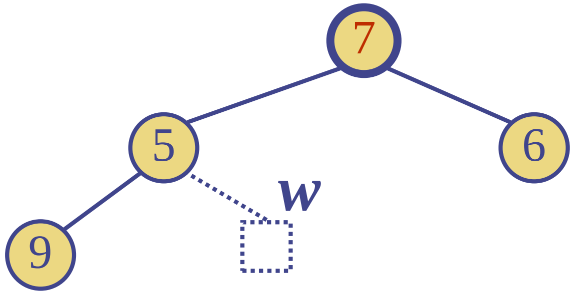
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node w
  - Remove w
  - Restore the heap-order property (discussed next)

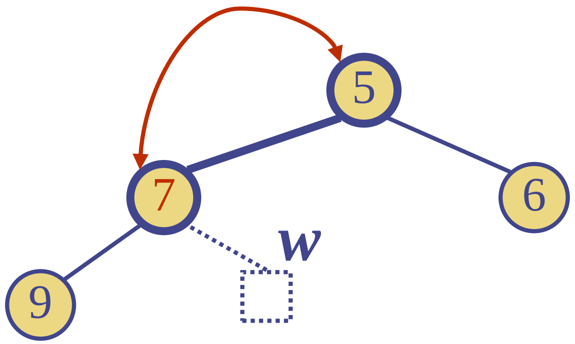




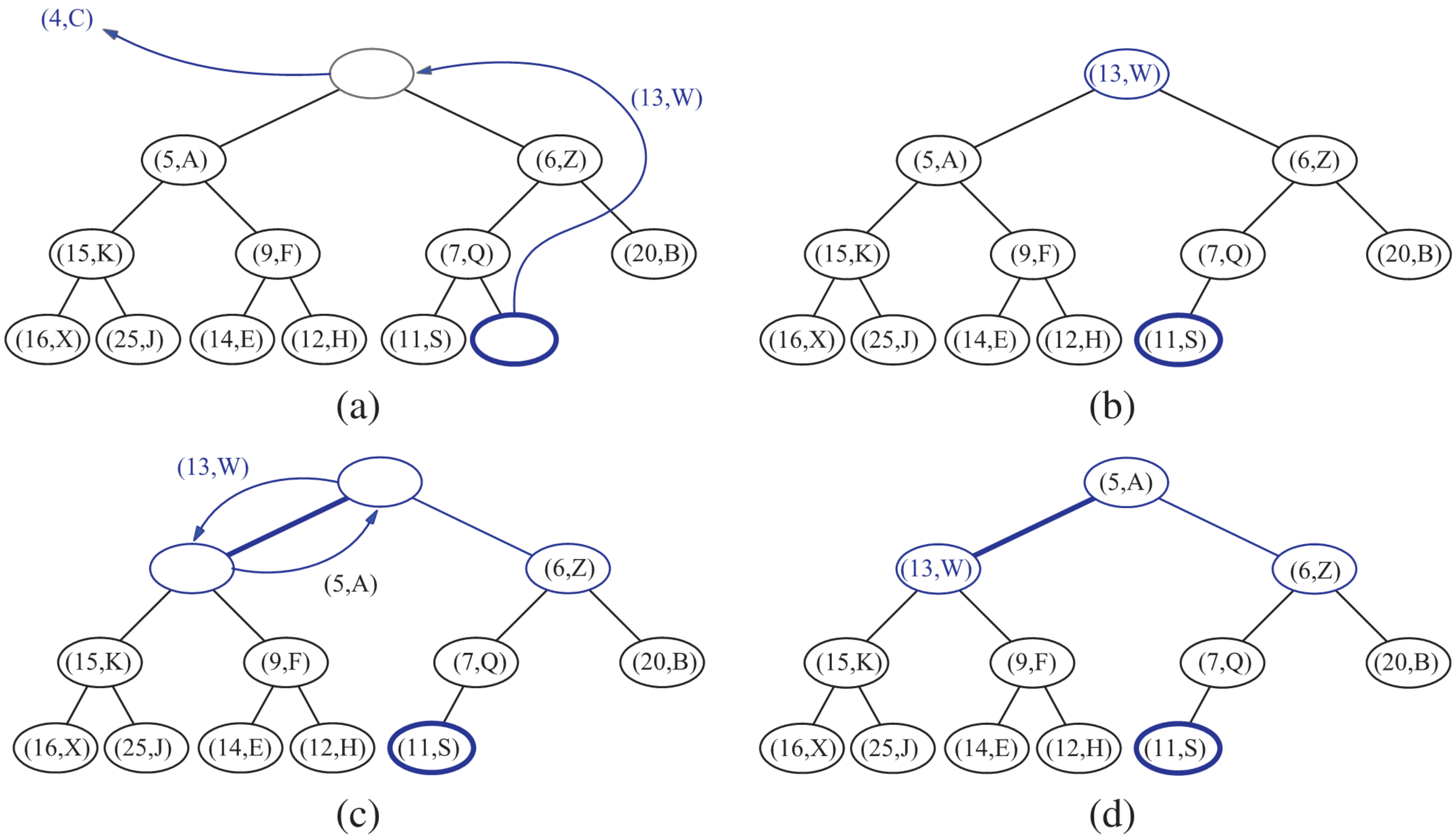
Downheap

- 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 (but which path?)
- Upheap 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

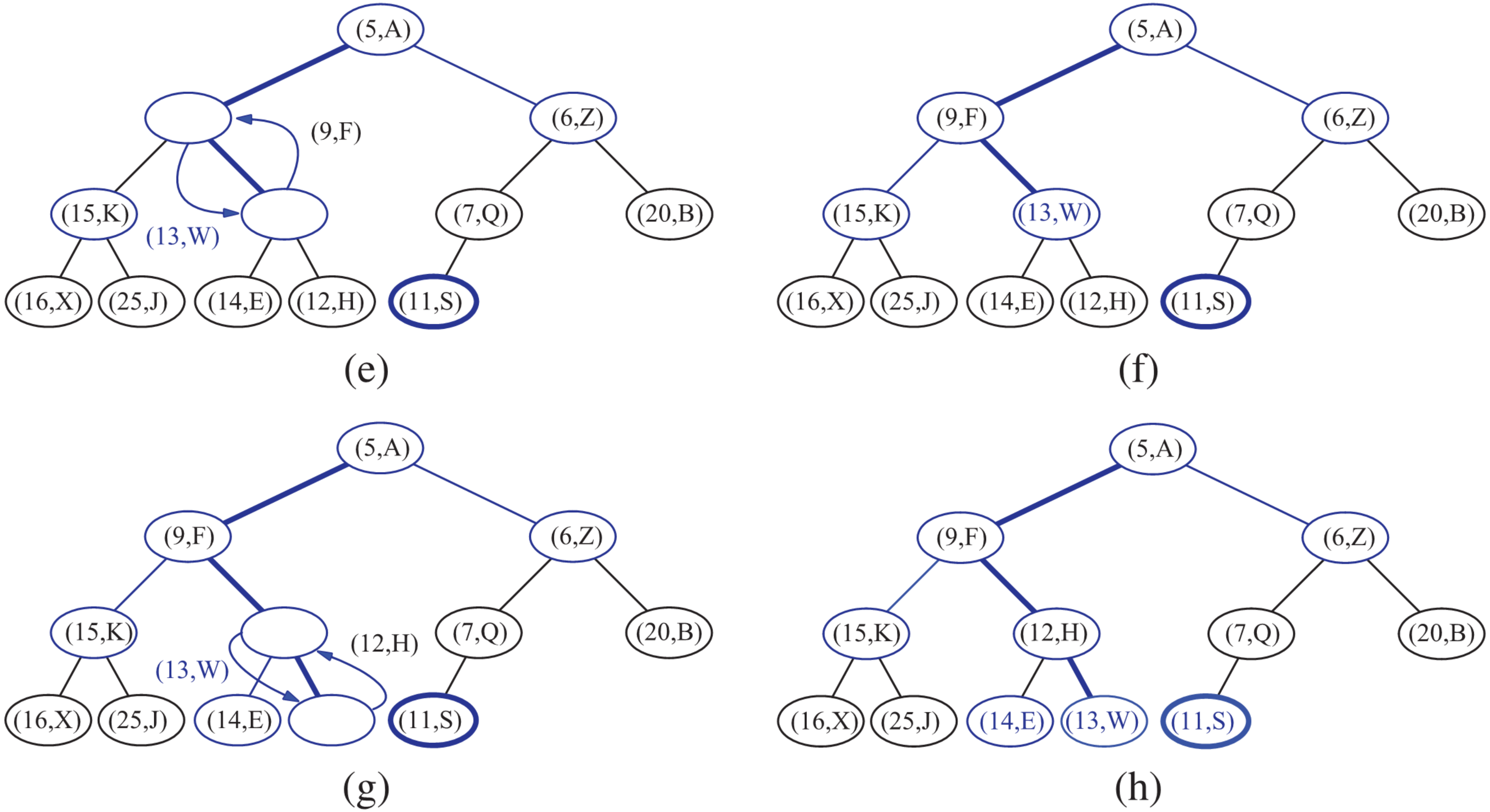




removeMin


(4,C)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(a)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(b)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(c)
(5,A)
(13,W)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(d)

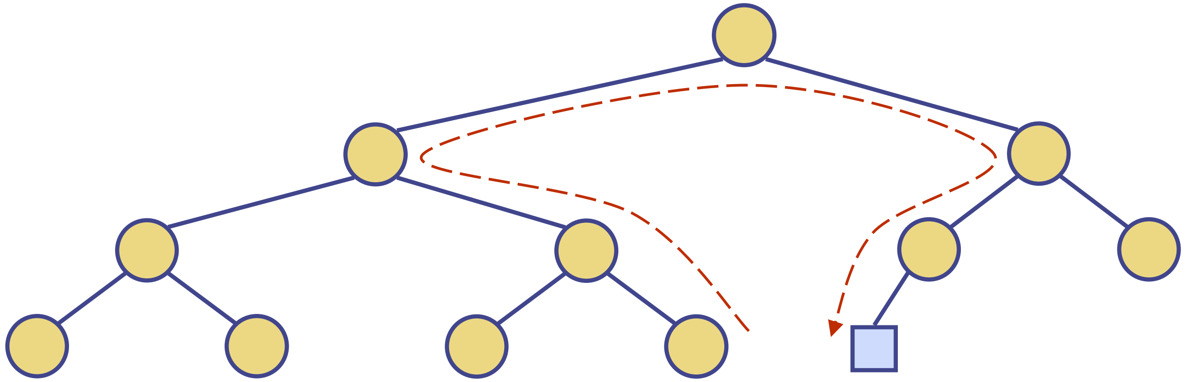
removeMin



Updating the Last Node

- How can we find the insertion node (a new last node)?
  - The insertion node can be found by traversing a path of $O(\log n)$ nodes
  - (1) Go up until a left child or the root is reached
  - (2) If a left child is reached, go to the right child
  - (3) Go down left until a leaf is reached

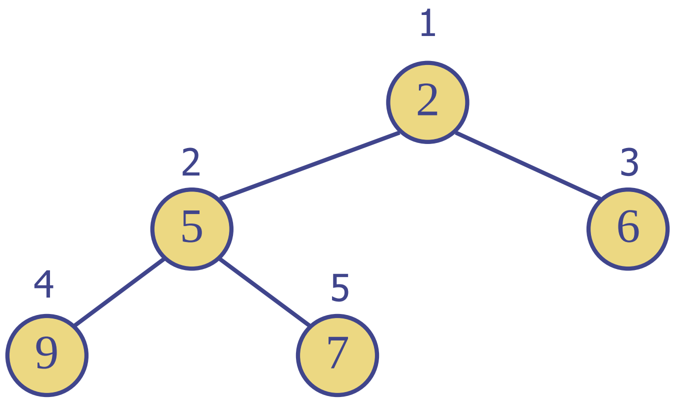
- Similar algorithm for updating the last node after a removal

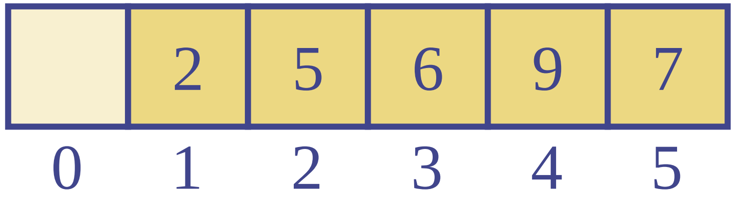


Vector-based Heap Implementation

IUT-ECE

- We can represent a heap with n keys by means of a vector of length $n + 1$
- For the node at rank i
  - the left child is at rank $2i$
  - the right child is at rank $2i + 1$
- Links between nodes are not explicitly stored
- The cell of at rank 0 is not used
- Operation insert corresponds to inserting at rank $n + 1$
- Operation removeMin corresponds to removing at rank n





List-based vs. Heap-based

List-based

Operation	*Unsorted List*	*Sorted List*
size, empty	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$
min, removeMin	$O(n)$	$O(1)$

Heap-based

Operation	*Time*
size, empty	$O(1)$
min	$O(1)$
insert	$O(\log n)$
removeMin	$O(\log n)$