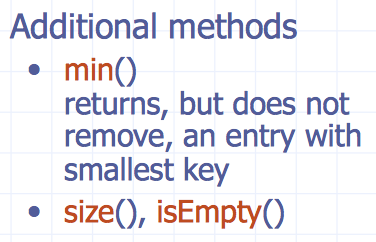
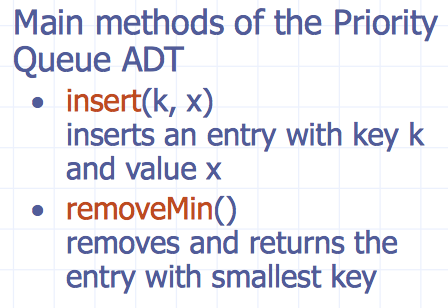
# Ch 9: Priority Queues

# 9.1 Priority Queue ADT

It stores a collection of entries.

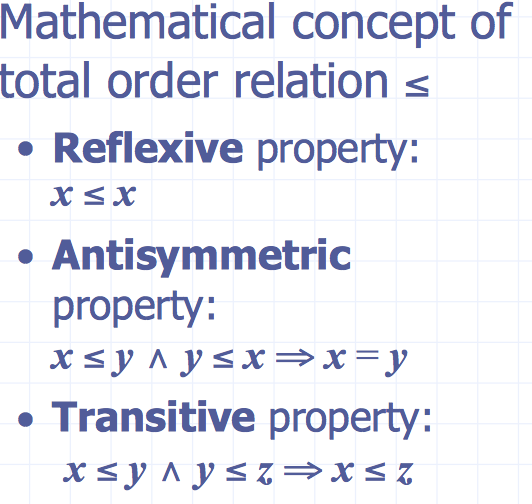
Each entre = (key,value)



# 9.2 Total Order Relation

Keys in a priority queue can be arbitrary objects on which an order is defined

2 distinct items in queue can have same key



## 9.2.1 Entry ADT

An entry in a priority queue is simply a key- value pair

Priority queues store entries to allow for efficient insertion and removal based on keys

Methods:

* getKey: returns the key for this entry
* getValue: returns the value associated with this entry

## Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 9.42.29 PM.png9.2.2 Comparator ADT

A comparator encapsulates the action of comparing two objects according to a given total order relation.

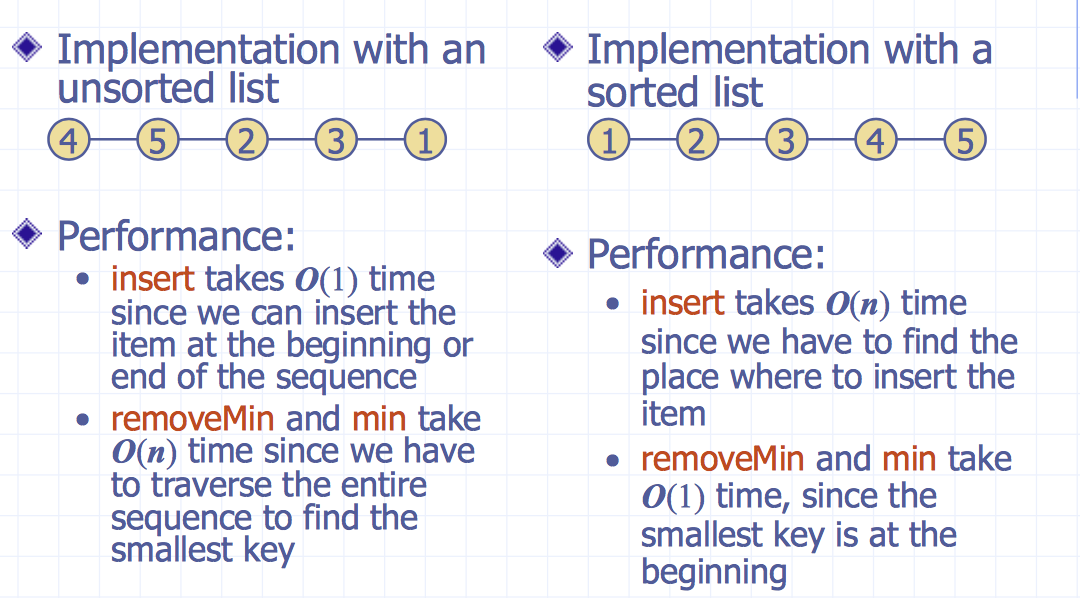
Comparator is external to the keys being compared

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 9.43.06 PM.png9.3 Sorting with a priority queue

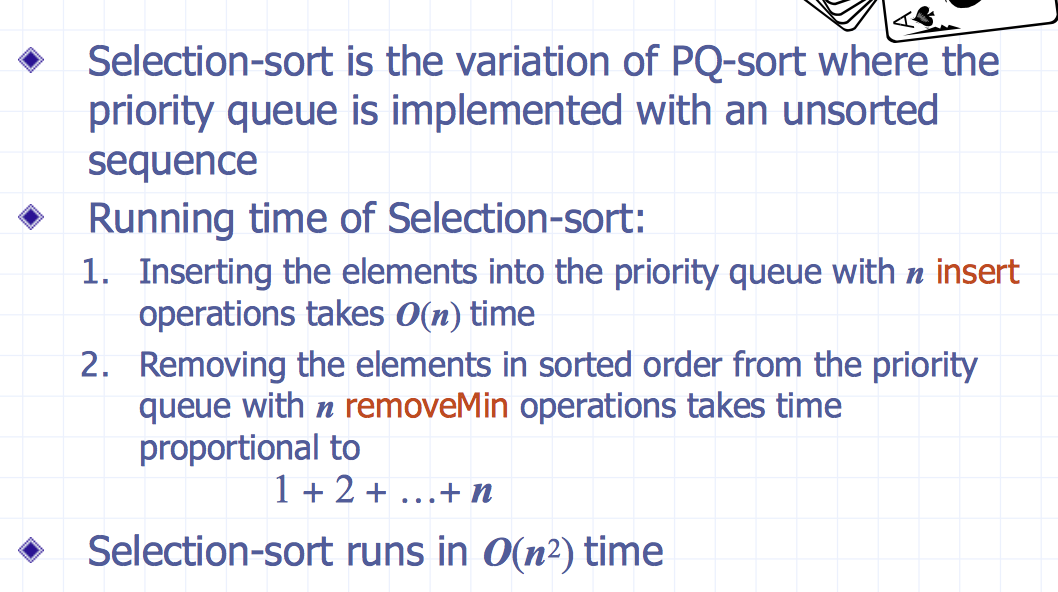
Use priority queue to sort set of comparable elements:

1. ***Insert*** elements ***one by one*** with a series of insert operations
2. **Remove** elements in sorted order with series of removeMin operations
3. Running time of this sorting method depends on priority queue implementation

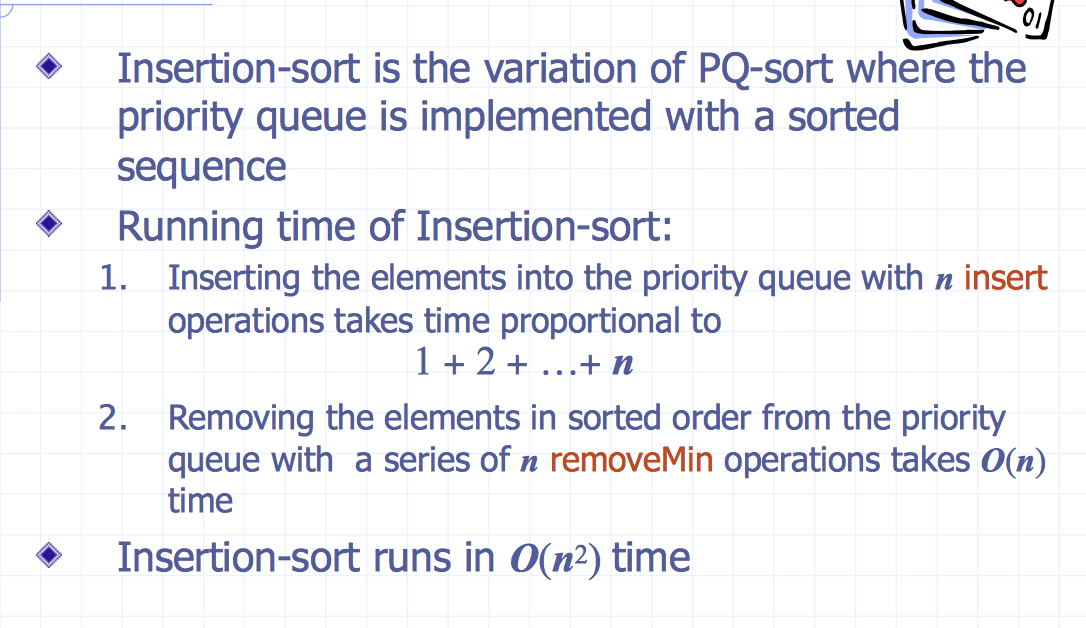
# 9.4 Sequence-based Priority Queue

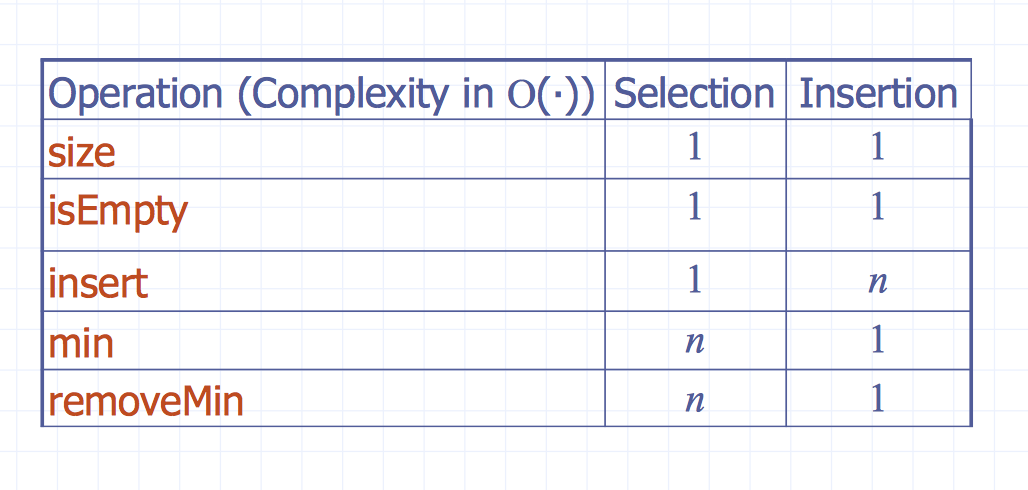


# 9.5 Selection-Sort

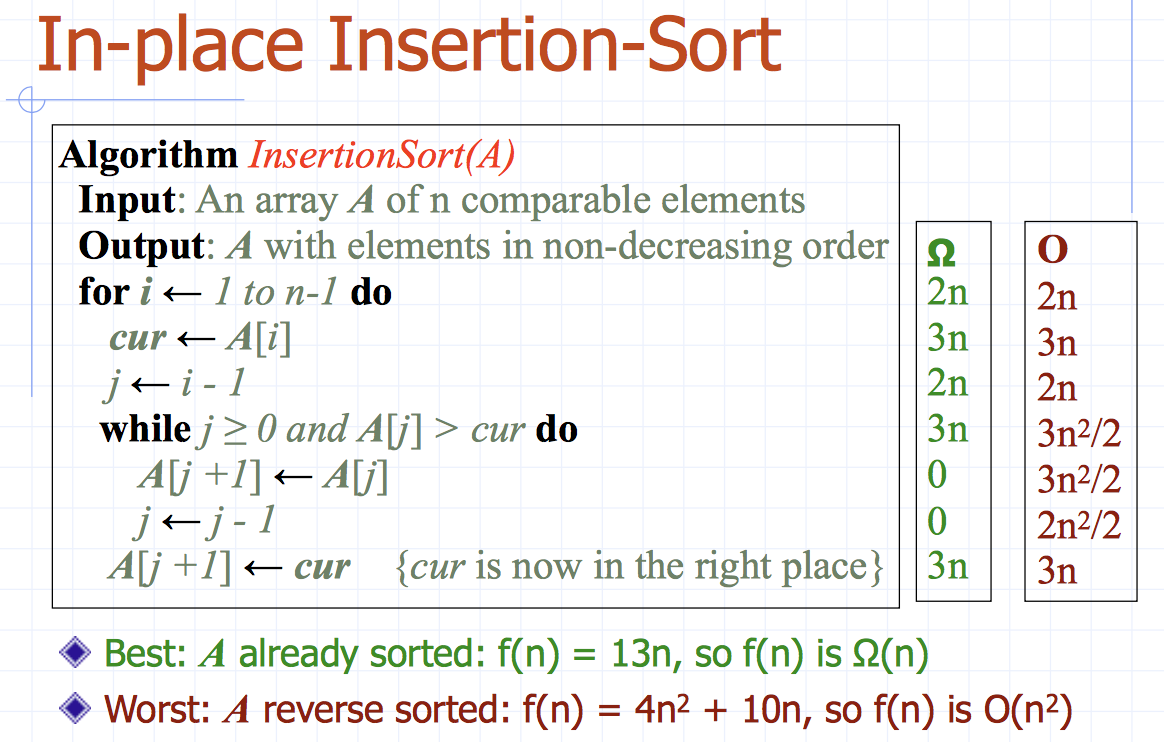
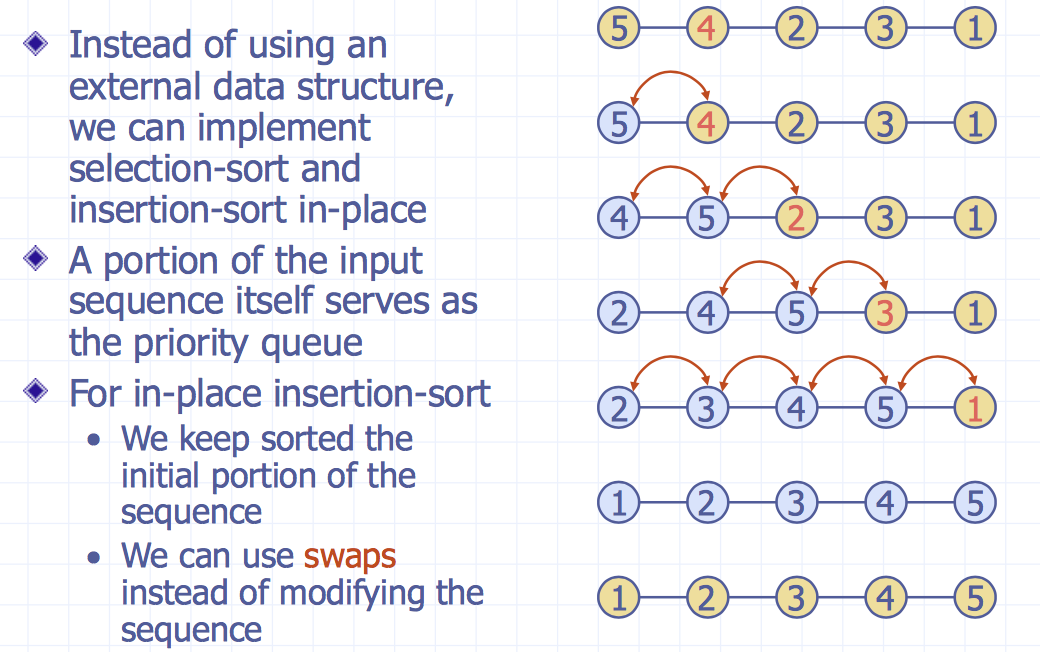


# 9.6 Insertion-Sort

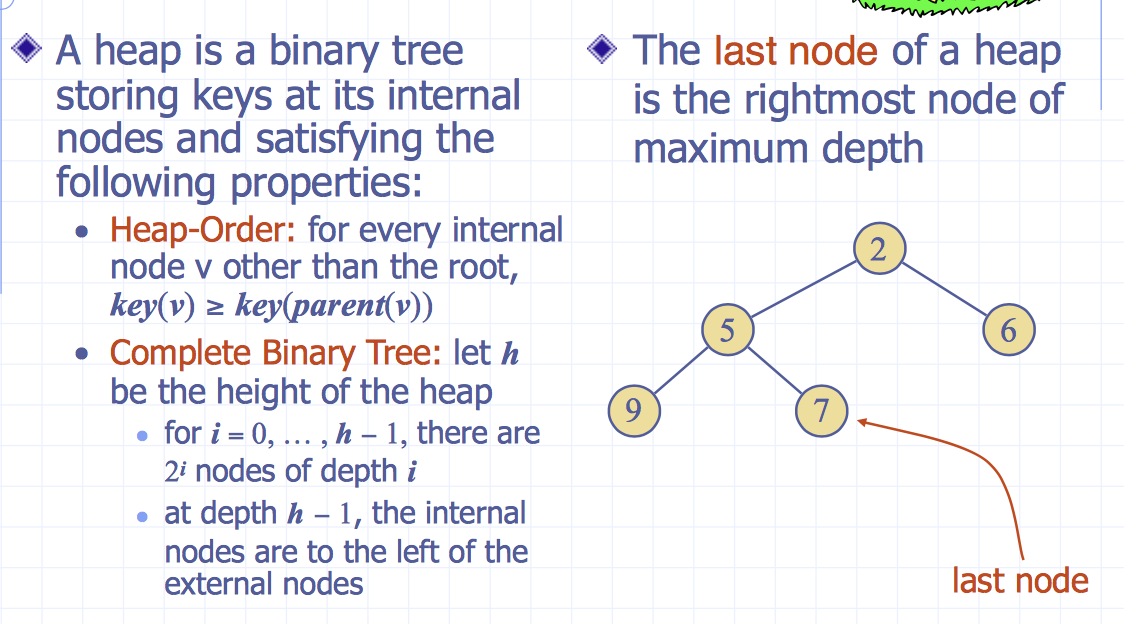




# 9.7 In-place Insertion-Sort



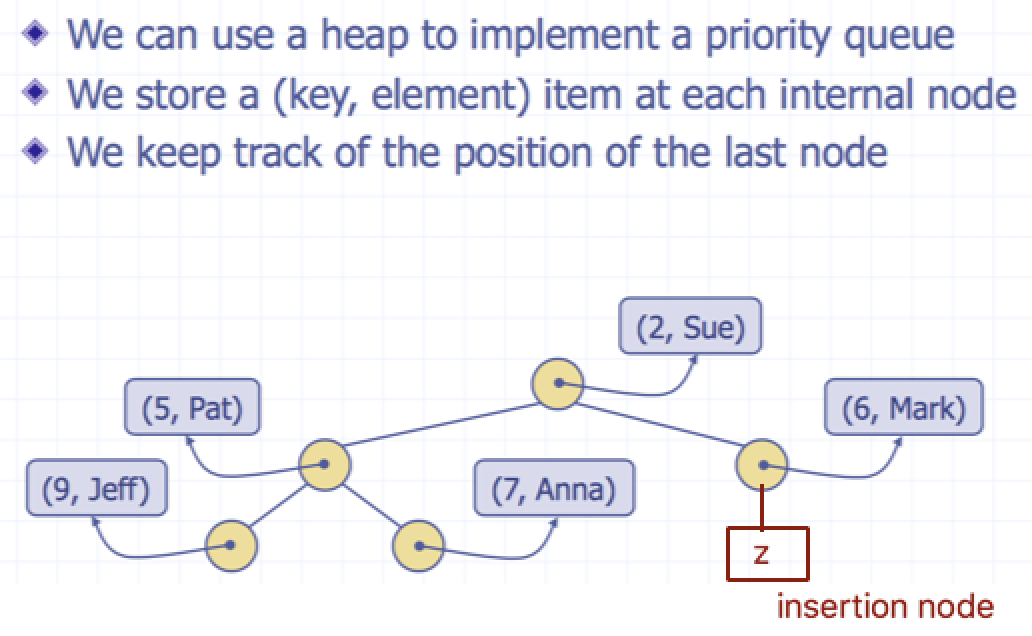
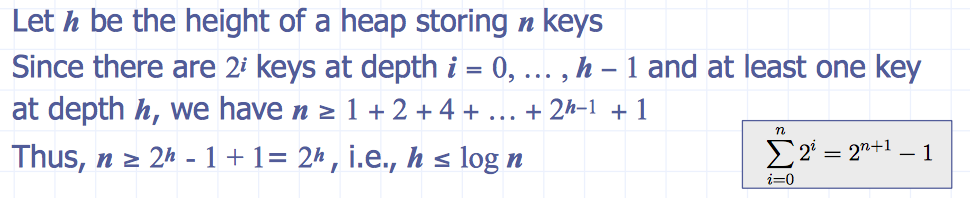
# 9.8 Heap



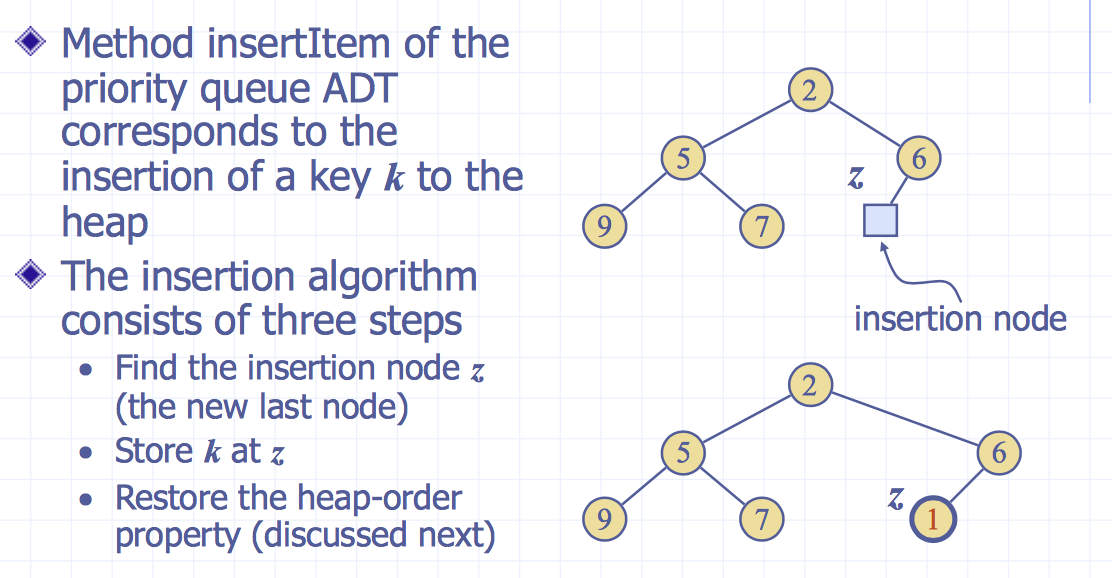
Complete binary tree : every level except last is full

### Theorem of height of heap:

A heap storing n keys has height O(log n)

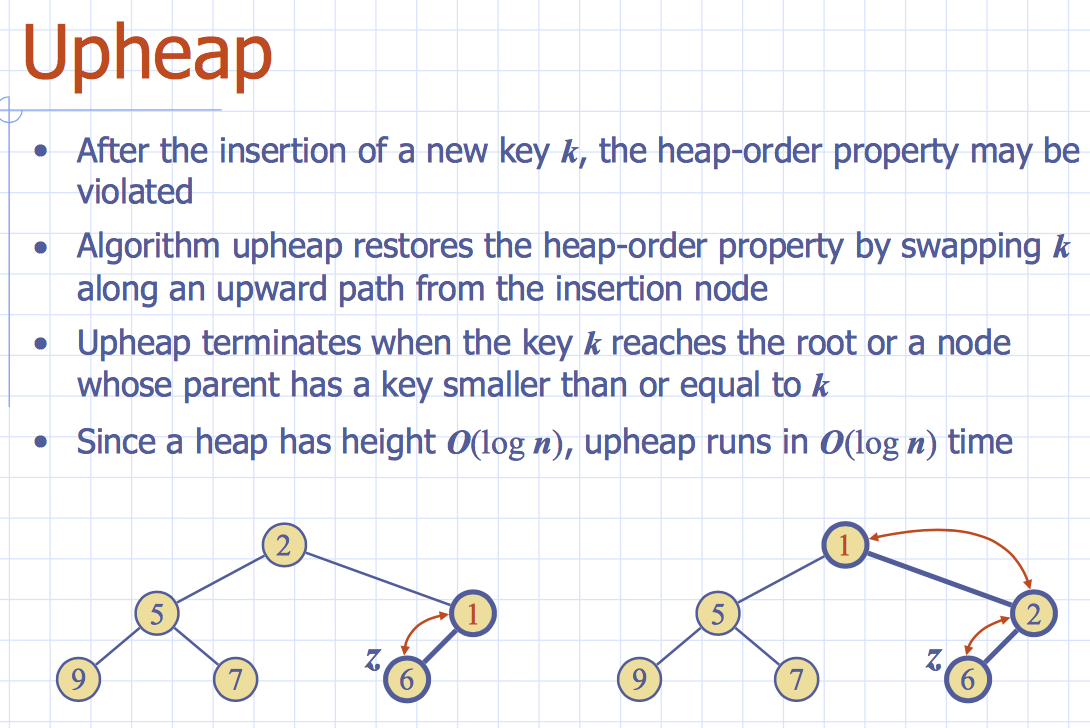
Proof: Heaps and Priority Queues

## 9.8.1 Insertion into a Heap



partial order key>= key of parent

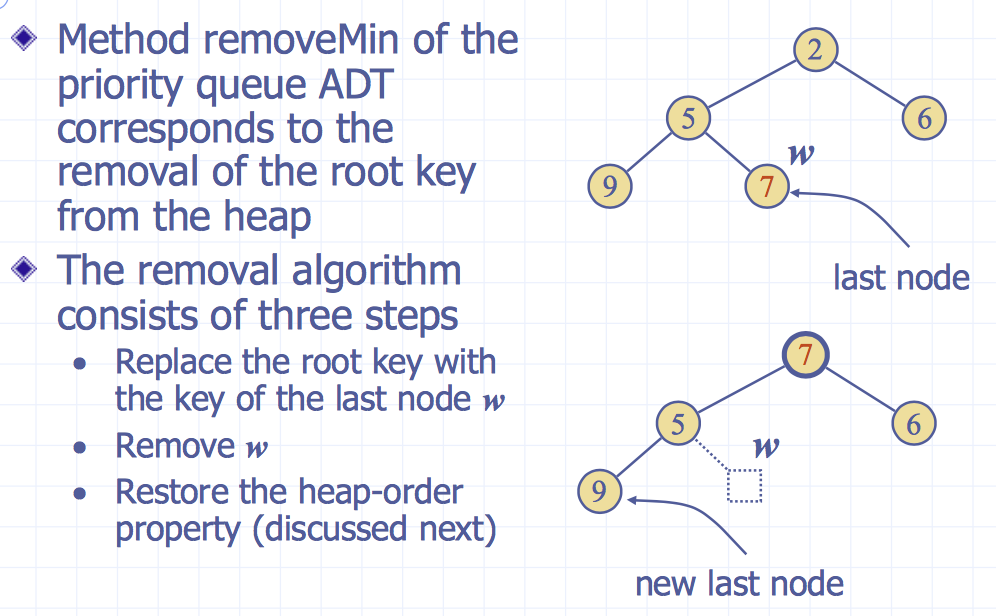
heap-order: follow to root -> becomes logn

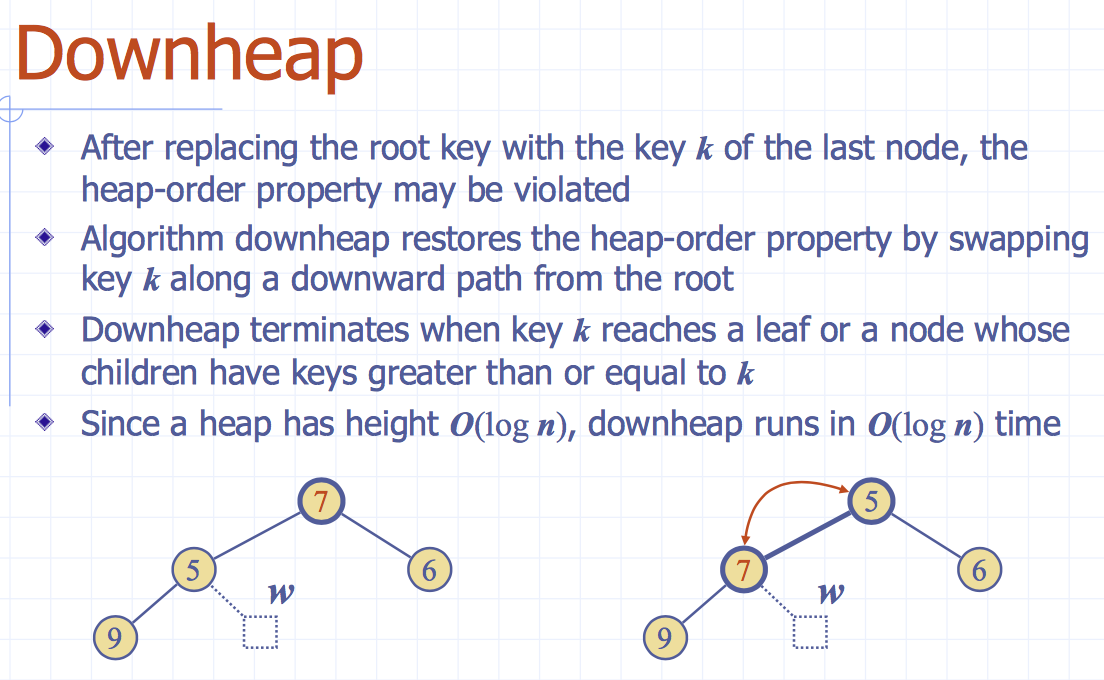


Become 1 is smaller than 6, swap them, swap until heap order restored

See example if needed

## 9.8.2 Removal from a Heap





### Which child is replaced in downheap?

If a node has ***no right child***, choose the ***left child***

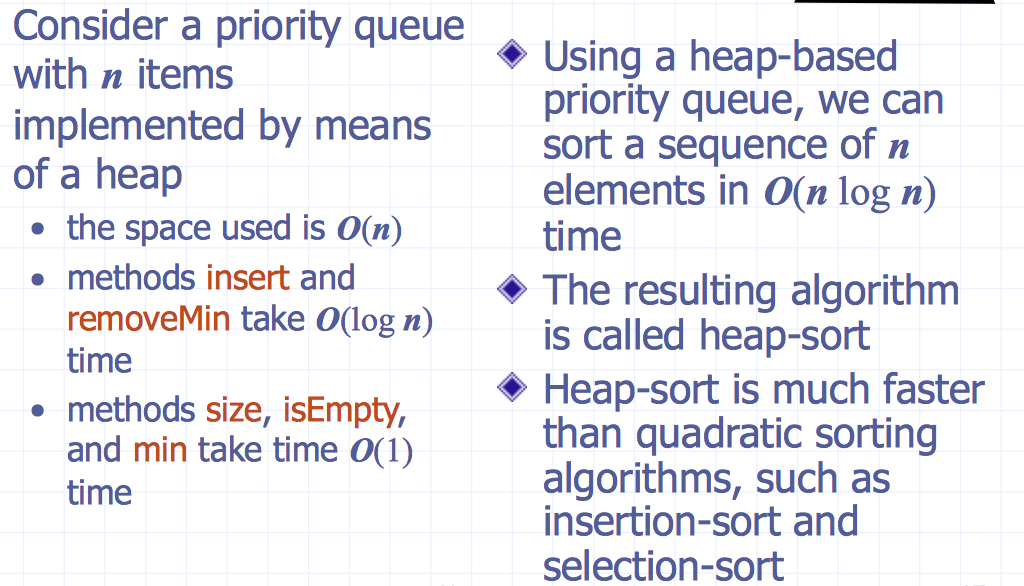
If a node has ***both children***, choose the ***one with the smallest key***

### 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 à
* Go down left until a leaf is reached

## 9.8.3 Heap-Sort



Can be implemented by dynamic tree or array base

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-21 at 11.50.54 AM.png9.9 Vector-based Heap Implementation

Can represent a heap with n keys by means of a vector of length n

For the node at rank i

* left child is at rank 2i+1
* right child is at rank 2i+2

***Links*** between nodes are ***not explicitly stored***

***insert*** = inserting at rank n + 1

***removeMin*** = removing at rank n (by replacing content of rank 0 with rank n)

Ex: n = 5 left child is 2(1)+1 = 3 and right child 2(1)+2 = 4

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-21 at 12.38.11 PM.png9.10 Merging Two Heaps

* Given 2 heaps and a key k
* We ***create new heap*** with ***root node*** storing **k** and with two heaps as subtrees
* We perform ***downheap*** ***to restore heap-order*** property

In this case = best case because both have same depth

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-21 at 12.40.07 PM.png9.11 Bottom-up heap construction

Construct ***heap storing n given keys*** in using a bottom-up construction with log n phases

In phase i, pairs of heaps with 2i −1 keys are merged into heaps with 2i+1−1 keys

Example:

