Chapter 6

Binary Search Trees

1st semester

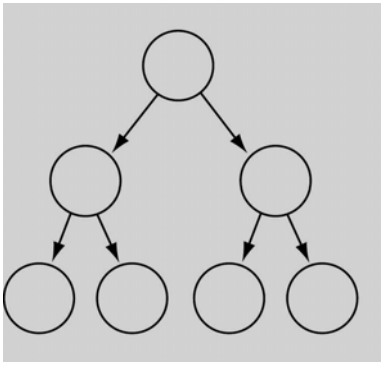
2007/2008

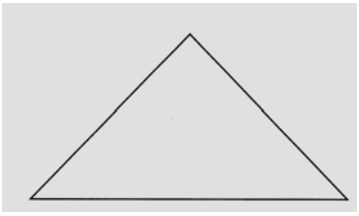
Instructor: Mamoun Nawahdah

Heaps

Full Binary Tree

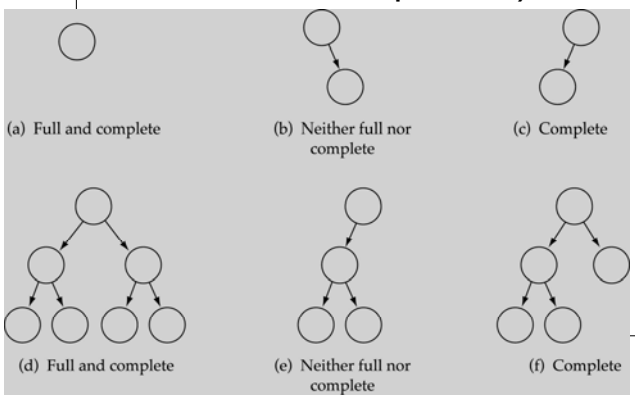
- Every non-leaf node has two children
- All the leaves are on the same level

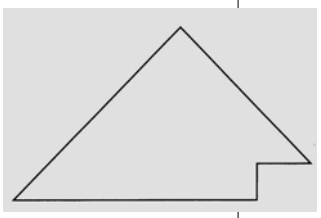




Complete Binary Tree

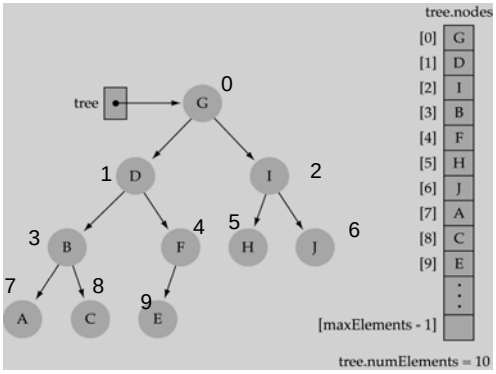
- A binary tree that is either full or full through the next-to-last level
- The last level is full from left to right (i.e., leaves are as far to the left as possible)





Array-based representation of complete binary trees

- Preserve parent-child relationships by storing the tree elements in the array

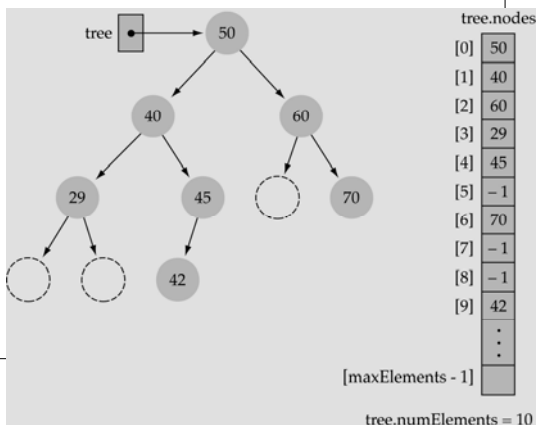


Array-based representation of complete binary trees

- Parent-child relationships:
  - left child of *tree.nodes[index] = tree.nodes[2*index+1]*
  - right child of *tree.nodes[index] = tree.nodes[2*index+2]*
  - parent node of *tree.nodes[index] = tree.nodes[(index-1)/2]*
    (int division-truncate)
- Leaf nodes:
  - *tree.nodes[numElements/2]* to *tree.nodes[numElements - 1]*

Array-based representation of complete binary trees

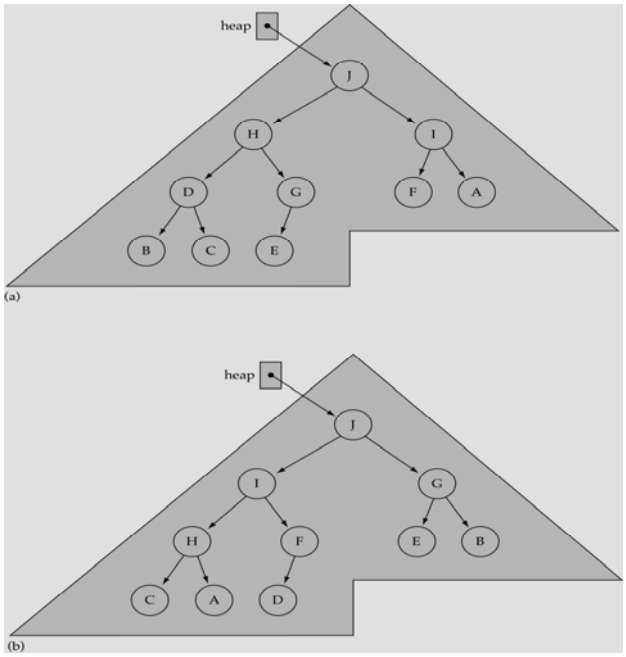
- Full or complete trees can be implemented easily using an array-based representation (elements occupy contiguous array slots)
- "Dummy nodes" are required for trees which are not full or complete



What is a heap?

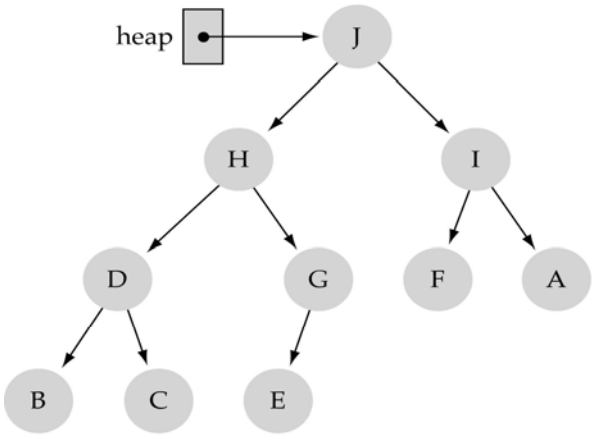
- It is a binary tree with the following properties:
  - *Property 1:* it is a complete binary tree
  - *Property 2:* the value stored at a node is greater or equal to the values stored at the children

What is a heap? (cont.)



Largest heap element

- From *Property 2*, the largest value of the heap is always stored at the root

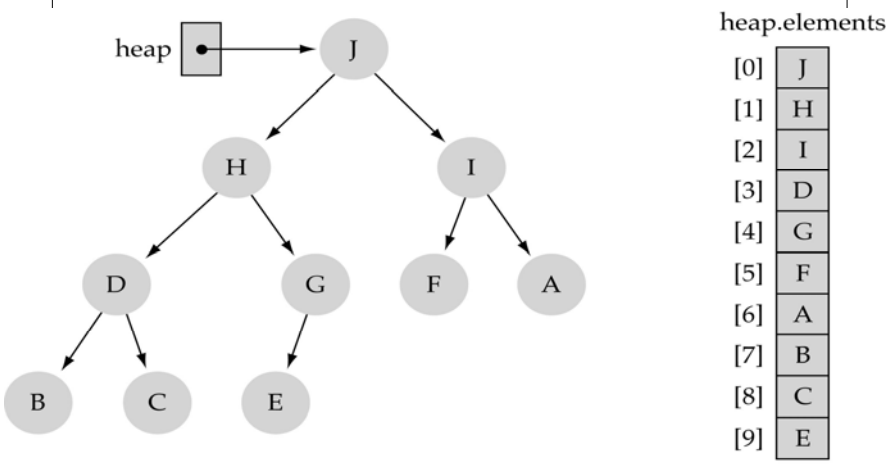


heap.elements

[0]	J
[1]	H
[2]	I
[3]	D
[4]	G
[5]	F
[6]	A
[7]	B
[8]	C
[9]	E

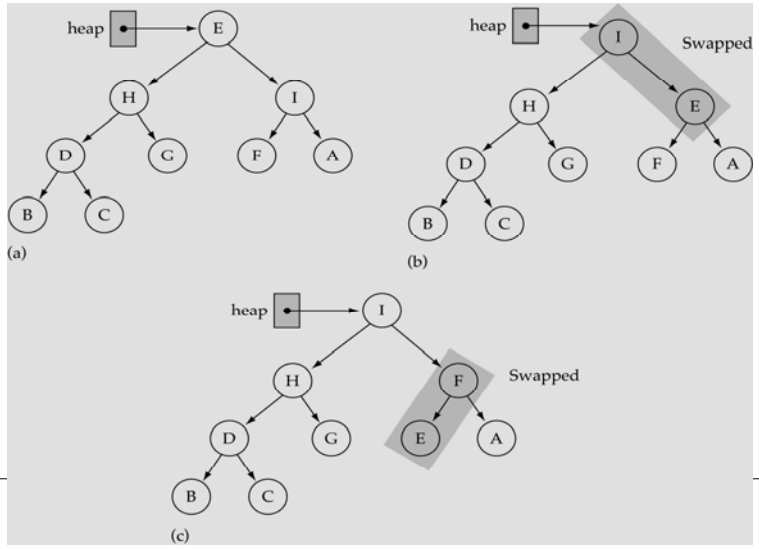
Heap implementation using array representation

- A heap is a complete binary tree, so it is easy to be implemented using an array representation



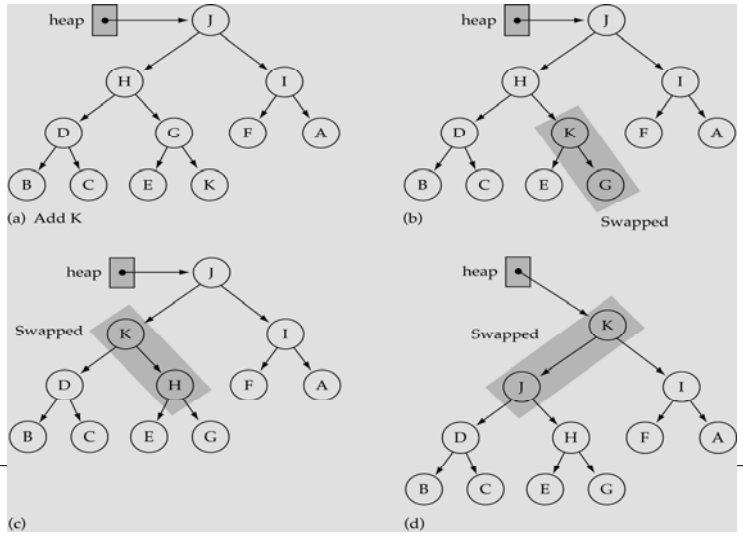
The ReheapDown function (used by deleteItem)

- Assumption: heap property is violated at the root of the tree



The ReheapUp function
(used by insertItem)

- Assumption: heap property is violated at the rightmost node at the last level of the tree



ReheapDown function

rightmost node in the last level

```
template<class ItemType>
void HeapType<ItemType>::ReheapDown(int root, int bottom) {
 int maxChild, rightChild, leftChild;

 leftChild = 2*root+1;
 rightChild = 2*root+2;

 if(leftChild <= bottom) {  // left child is part of the heap
   if(leftChild == bottom) // only one child
     maxChild = leftChild;
   else {
     if(elements[leftChild] <= elements[rightChild])
       maxChild = rightChild;
     else
       maxChild = leftChild;
   }
   if(elements[root] < elements[maxChild]) {
     Swap(elements, root, maxChild);
     ReheapDown(maxChild, bottom);
     }
  }
}
```

ReheapUp function

Assumption:
heap property
is violated at bottom

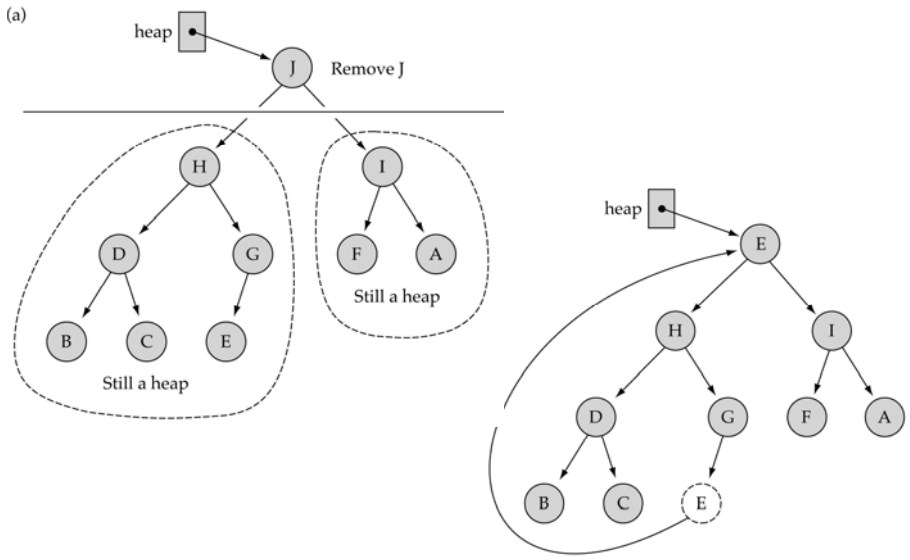
```
template<class ItemType>
void HeapType<ItemType>::ReheapUp(int
   root, int bottom){
 int parent;

 if(bottom > root) { // tree is not empty
   parent = (bottom-1)/2;
   if(elements[parent] < elements[bottom]) {
     Swap(elements, parent, bottom);
     ReheapUp(root, parent);
   }
 }
```

Removing the largest element from the heap

1) Copy the bottom rightmost element to the root
2) Delete the bottom rightmost node
3) Fix the heap property by calling *ReheapDown*

Removing the largest element from the heap (cont.)

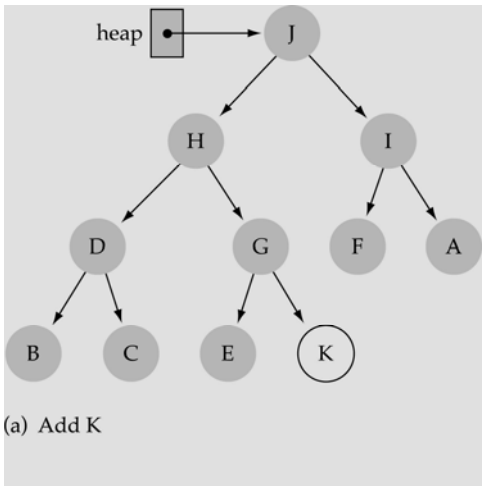


Inserting a new element into the heap

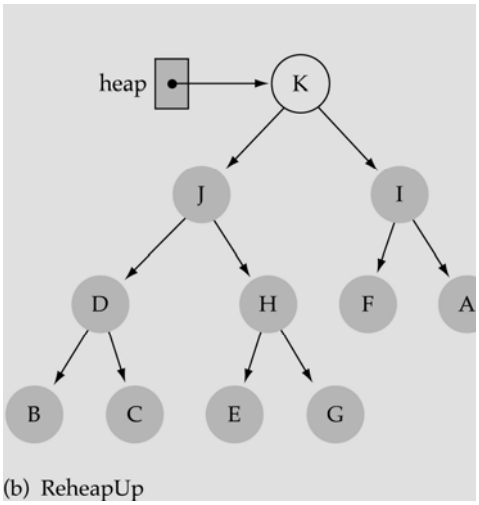
1) Insert the new element in the next bottom leftmost place

2) Fix the heap property by calling *ReheapUp*

Inserting a new element into the heap (cont.)



(a) Add K



(b) ReheapUp

Priority Queues

- What is a priority queue?
  - It is a queue with each element being associated with a "priority"
  - From the elements in the queue, the one with the highest priority is dequeued first