

# Data Structures G5029

## Lecture 6

Kingsley Sage  
Room 5C16, Pevensey III  
[khs20@sussex.ac.uk](mailto:khs20@sussex.ac.uk)

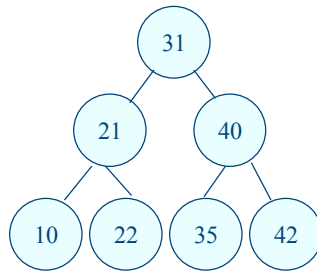
© University of Sussex 2006

## Lecture 6

- Binary heaps and Priority Queues

## Binary search trees

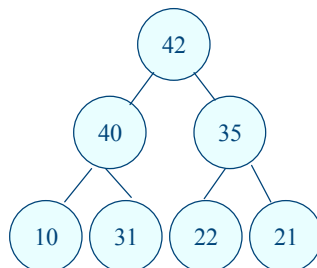
- Remember binary search trees?.



- The items are placed in the tree such that the value stored at each node is greater than the values stored in its left sub-tree, and less than the value stored in its right sub-tree.
- You can think of the items as being ordered from left to right.
- We obtain an interesting alternative structure if the items are ordered from bottom to top ...

## Towards the binary heap ...

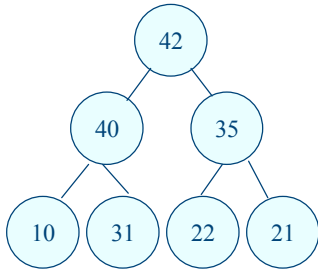
- Ordered from bottom to top ...



- The defining property is now that the value stored at each node of the tree is greater than or equal to the values stored in both sub-trees.
- A simple test for whether this property holds is that the value stored at every node should be no greater than the value stored at its parent (apart from the root which doesn't have a parent). It follows that every path from the root to a leaf forms a decreasing sequence.

## Towards the binary heap ...

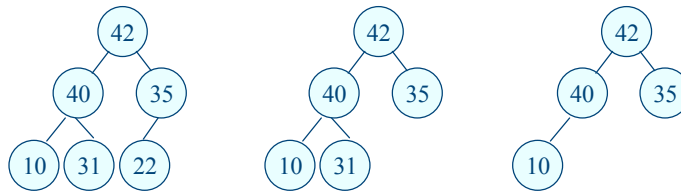
- The idea of being well-balanced ...



- Trees satisfying the defining property are particularly useful if they are well-balanced, in the sense of having short path lengths from root to leaves
- They don't have to be quite as well-balanced at this tree ...

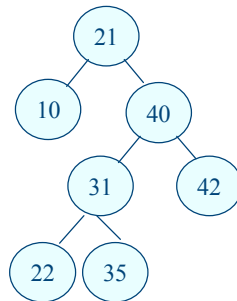
## Towards the binary heap ...

- Trees like these are OK ...



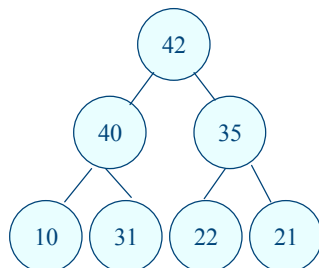
## Towards the binary heap ...

- But not a tree like this, where there are some paths from leaf to the root whose length differ by more than 1 ...



## The binary heap

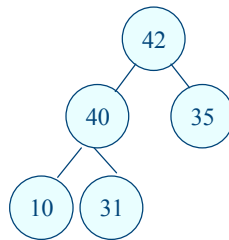
- Ordered from bottom to top ...



- It is also very convenient if the bottom level is filled from left to right (or right to left, we just need to agree on one or the other). Binary trees of this type are called **complete**.
- Complete trees are defined as being completely filled, except possibly for the bottom level, which is filled from left to right.
- A complete binary tree that satisfies the ordering property is called a **binary heap**.

## Array representation

- A complete binary tree has a simple array representation. Suppose we number the nodes from left to right, beginning at the top and ending at the bottom. Then we can store the various data items in the corresponding elements of an array. For example ..



42	40	35	10	31			
0	1	2	3	4	5	6	7

- This in fact corresponds to the low level enumeration of the tree.
- Note that we only use an initial segment of the array. Provided the array is long enough, and we know the number of tree nodes, it doesn't matter how many unused elements there are at the end.

## Arithmetic of tree traversal

- An important benefit of the array representation is that we can move around the tree, from parent to child and child to parent, by simple arithmetic.
  - Children of node  $i$  are at  $2i+1$  and  $2i+2$
  - Parent of node  $i$  is at  $(i-1)/2$
  - Node  $i$  may not have any children, or it may have only one child
  - Node  $i$  has a left child if  $2i+1 < n$  and a right child if  $2i+2 < n$
  - Node  $i$  has a parent if  $i > 0$ , otherwise it is the root
- Where division to obtain the parent returns the integer part.
- Be careful if you're looking at a text where arrays are not zero-indexed.

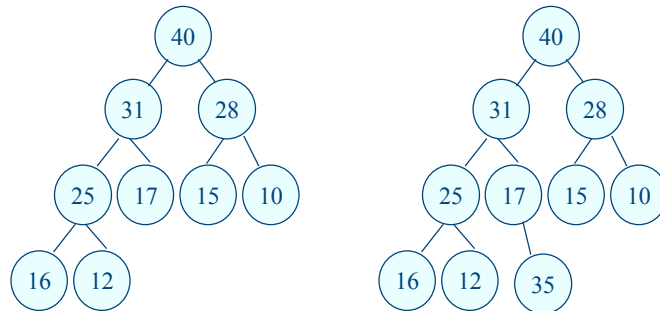
## Priority Queues

- OK – so what do we use these binary heaps for?
- Two important applications:
  - Priority queues
  - Sorting
- Priority queues:
  - We modify our previous queue concept by attaching a priority to items in the queue. For example, jobs waiting in a printer queue, processes awaiting execution on a processor, passengers waiting for an airline flight etc ...
  - If an item of a higher priority joins the queue, it can jump to the front or, at least, progress to some place closer to the front on merit. The items in the queue are ranked.
  - We want the highest rank item to be served first.

## Implementation

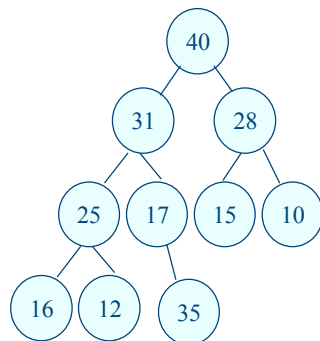
- If we use a binary heap to implement a priority queue, an item of highest priority will always be at the root of the tree.
  - This makes it very easy to determine the next item to emerge from the queue.
- Once we have removed that item, we must re-organise the heap so that an item of next highest priority amongst the remainder moves into the root position.
- Removal of an item must leave the heap ordered.
- When an object is inserted, it must be placed in the correct position so that the ordering property remains satisfied.
- Code level details are in the on-line course notes.

## Insertion algorithm



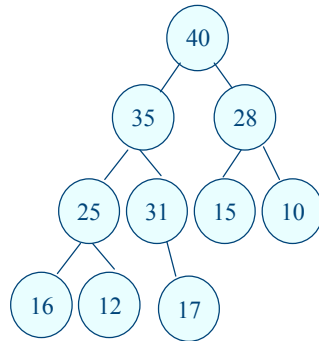
For a heap of integers, let's say that we want to insert 35. Since we must fill each layer from left to right, we add it as shown on the right hand side tree ...

## Insertion algorithm



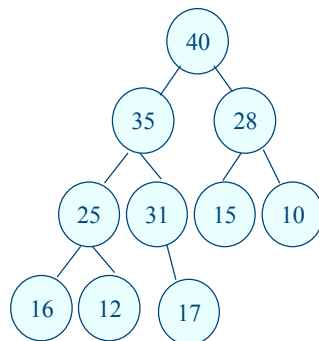
- But this now violates the ordering property.
- There is a simple algorithm for restoring the ordering. We simply "bubble up" the 35 up the tree until the order of the path of the path from root to leaf is correct.
- This involves a sequence of transformations:
  - $(40,31,17,35) \rightarrow (40,31,35,17) \rightarrow (40,35,31,17)$
- Making these two exchanges, we get ...

## Insertion algorithm



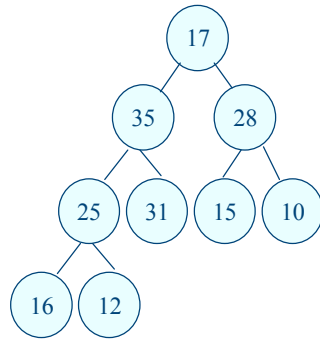
- The 35 has been promoted to its proper place.
- Note that the 35 is no less than its left descendant 25.
- By construction, the 35 had to be no less than its right descendant 31, otherwise we should not have made the final promotion.
- But the 31 had to be no less than the 25 in the first place, since the heap was originally well ordered.

## Removal algorithm



- Removing an item from a priority queue is simple as the next item to leave the queue will always be at the top of the heap.
- The only problem is that now we shall have an empty place.
- Since the heap must form a complete binary tree, something must fill this place.
- Furthermore, the bottom right position must be vacated, since there will now be one less item in the heap.
- So, to start the process, remove the 40 from the top and put the bottom right item at the now vacant top of the heap ...

## Removal algorithm



- This is now a complete binary tree, but it's not a heap because the 17 is smaller than both its children.
- The solution is to demote the 17 from the top, similar to the way we promoted items from the bottom for insertion.
- For promotion, we only had to consider a single parent, for demotion, we need to consider both children.
- We exchange the 17 with the larger of its two children (if they are the same, it doesn't matter which).
- And keep on going down the tree until we get to a node where the heap ordering is once again established, or we reach the bottom of the tree.

## Strict priority trees

- What should happen if two or more of the items in the queue have the same priority?
- One obvious answer would be to deal with the one that arrived first.
- Depends on the semantics associated with the task that we are using the queue to represent.
- Can add additional fields inside the node to capture information to implement such strict priority queues e.g. time variables, alphabetic order, whatever ...

## Next time ...

- OK, now we know what a binary heap is and how to insert and remove items to and from it.
- Next time, we look at how binary heaps can be used as the basis of an efficient sorting mechanism – heap sort.