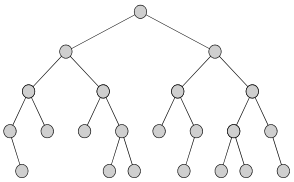
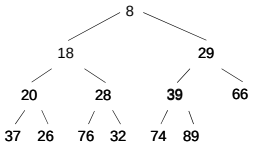


CSIS 3103

Ch 6: Priority Queues

Implementing a Heap

- A complete binary tree can be implemented efficiently using an array
- First element stores a reference to the root data



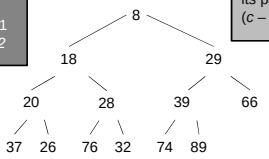
0	1	2	3	4	5	6	7	8	9	10	11	12
8	18	29	20	28	39	66	37	26	76	32	74	89

Implementing a Heap

For a node at position *p*,

L. child position: $2p + 1$
R. child position: $2p + 2$

For node at position *c* its parent is at $(c - 1) / 2$



0	1	2	3	4	5	6	7	8	9	10	11	12
8	18	29	20	28	39	66	37	26	76	32	74	89

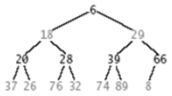
Parent, L. Child, R. Child

Inserting into a Heap

1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	18	29	20	28	39	66	37	26	76	32	74	89	8

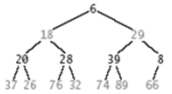
Parent, Child



2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	18	29	20	28	39	8	37	26	76	32	74	89	66

Parent, Child

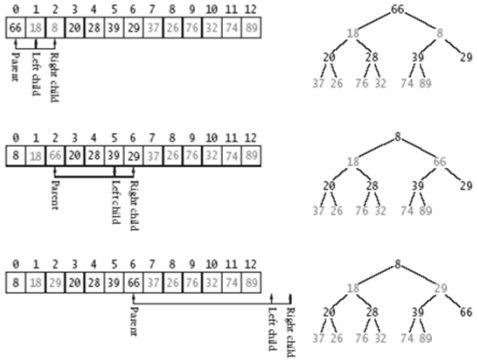


3.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	18	8	20	28	39	29	37	26	76	32	74	89	66

Parent, Child

Deleting from a Heap



Priority Queues

- Another kind of waiting line but not FIFO like a queue
- In a *priority queue*, only the highest-priority item is accessible
- A heap is used to implement a priority queue

Insertion into a Priority Queue

The `PriorityQueue` Class

- Java provides a `PriorityQueue<E>` class that implements the `Queue<E>` interface
- `peek`, `poll`, and `remove` methods return the *smallest* item in the queue

Method	Behavior
`boolean offer(E item)`	Inserts an item into the queue. Returns **`true`** if successful; returns **`false`** if the item could not be inserted.
`E remove()`	Removes the smallest entry and returns it if the queue is not empty. If the queue is empty, throws a `NoSuchElementException`.
`E poll()`	Removes the smallest entry and returns it. If the queue is empty, returns **`null`**.
`E peek()`	Returns the smallest entry without removing it. If the queue is empty, returns **`null`**.
`E element()`	Returns the smallest entry without removing it. If the queue is empty, throws a `NoSuchElementException`.

Using a Heap to Implement a Priority Queue

- In a priority queue, just like a heap, the smallest item always is removed first
- Heap insertion and removal is O(log *n*)
- The `java.util.PriorityQueue` uses an `Object[]` array

KWPriorityQueue Class

Data Field	Attribute
`ArrayList<E> theData`	An `ArrayList` to hold the data.
`Comparator<E> comparator`	An optional object that implements the `Comparator<E>` interface by providing a `compare` method.
Method	**Behavior**
`KWPriorityQueue()`	Constructs a heap-based priority queue that uses the elements' natural ordering.
`KWPriorityQueue (Comparator<E> comp)`	Constructs a heap-based priority queue that uses the `compare` method of `Comparator comp` to determine the ordering of the elements.
`private int compare(E left, E right)`	Compares two objects and returns a negative number if object `left` is less than object `right`, zero if they are equal, and a positive number if object `left` is greater than object `right`.
`private void swap(int i, int j)`	Exchanges the object references in `theData` at indexes `i` and `j`.