**QUEUES**

Queues

* First In First Out (remember stacks = L I F O)

Characteristic Operations:

* **Enqueue(Queue q, Item it)** add item onto queue
* **Item it = dequeue(Queue q)** remove item from stack

Other possible operations:

* **isEmpty(Queue q)** queue contains no items
* **Length(Queue q)** how many items in queue
* **Show(Queue q)** display queue on stdout
* **Queue q = newQueue()** create new empty queue
* **dropQueue(Queue q)** release queue data

Queue implementations

* As a linked list w/ array data
  + Pointer to first element of the list
  + Pointer to last element of the list
  + Dequeue = take off FIRST element
  + Enqueue = add to LAST element

**PRIORITY QUEUES**

Instead of FIFO, it is “one with the highest priority goes out” regardless of what goes in first.

* Items are processed in order of “key” / importance

Priority Queues (PQs) provide this via:

* **void join(PQ, Item)**: insert item in PQ
* **Item leave(PQ)**: remove item with the highest priority
* **Item remove(PQ, Key)**: remove item with specified key (non-standard)

Plus generic ADT operations:

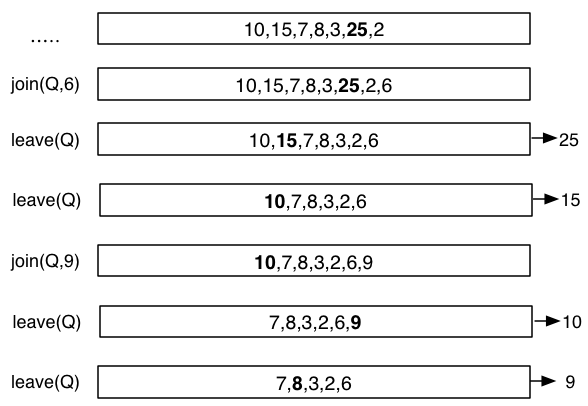
* **newPQ** / **dropPQ** / **isEmpty**…

Highest priority could be:

* largest key
* smallest key
* E.g. taking out item with the highest or lowest timestamp

Priority order may involve “weight” based on other factors than just a key

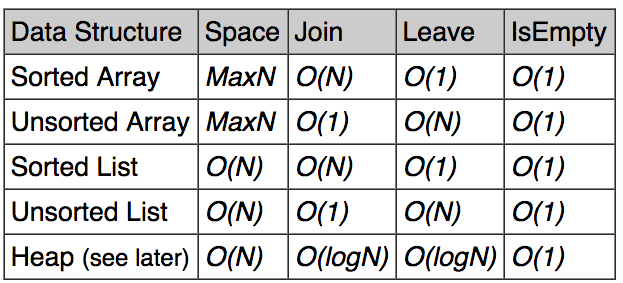
Behaviour of priority queue



Highest priority = largest no.

* 6 joins queue
* Highest (25) leaves queue
* Highest (15) leaves queue
* 9 joins queue
* Highest (10) leaves queue
* Highest (9) leaves queue

Priority Queue Cost Representations:



For a **PQueue** containing N items.

* Space = number of items in the data structure

HEAP is one of the better queue representations

UNSORTED is simple to join  
SORTED is simple to leave

**RECURSION**

**Add notes here**

**LISTS**

**Add notes here**

**GRAPH FUNDAMENTALS**