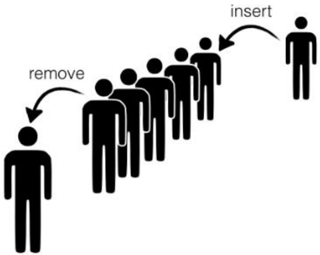
Priority Queue ADT

Lecture 02.05
by Marina Barsky

Recap: Queue ADT

A ***queue*** is an abstract data type supporting the following main operations:

- *enqueue* (e) adds an element to the back of the queue
- *dequeue* () extracts an element from the front of the queue

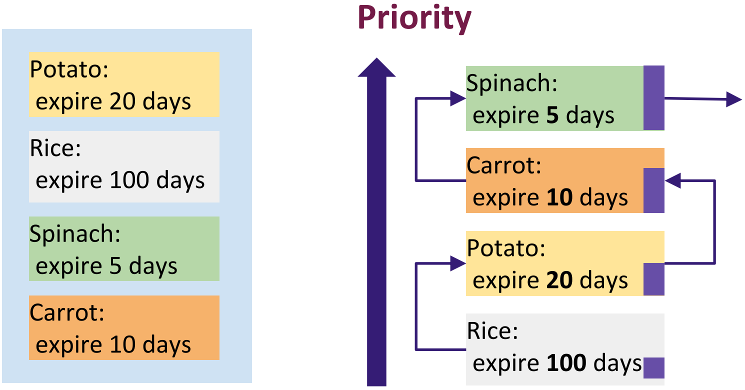


➢ The order in which elements go out of the queue is defined by the time at which they were added to the queue

Priority queue

- A ***priority queue*** is a generalization of a *queue* where each element is assigned a priority and elements come out in order of priority
- If the priority is the earliest time they were added to the queue then priority queue becomes a regular queue
- We are interested in a case when priority of each element is not related to the time when the element was added to the queue

Example



- Add items in order of purchasing
- Consume the items by priority: the items that expire soon have higher priority

Specification

Priority Queue is an **abstract data type** supporting the following main operations:

- ➔ ***insert(e,p)**** - adds a new element with priority *p*
- ➔ ***top()*** **-** gives an element with the highest priority
- ➔ ***pop()*** - removes and returns the element with the highest priority

---

*To simplify the discussion we use *insert(e),* where *e* is a number which reflects the priority

Example

Storage:

Operations:

Example

Storage:

Operations:

insert(5)

Example

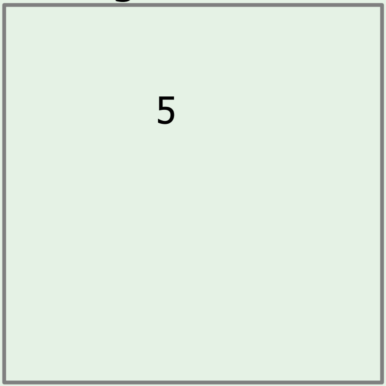
Storage:

5

Operations:

Example

Storage:

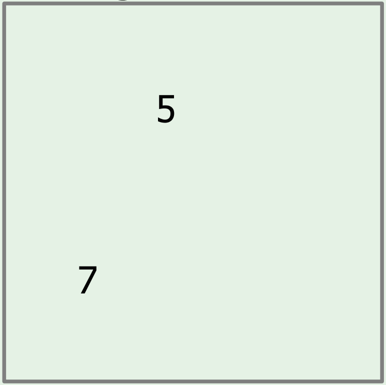


Operations:

insert(7)

Example

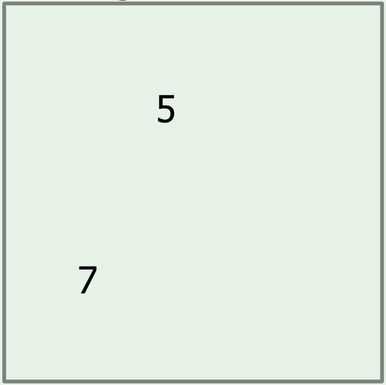
Storage:



Operations:

Example

Storage:

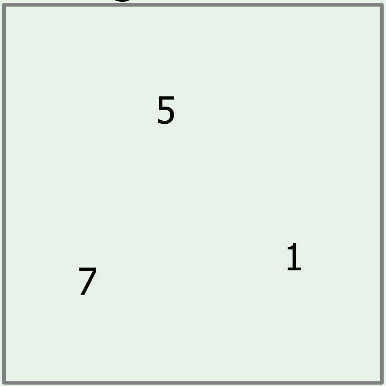


Operations:

insert(1)

Example

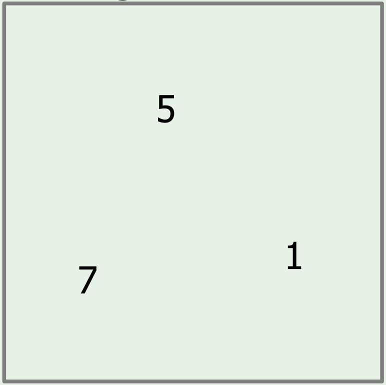
Storage:



Operations:

Example

Storage:



Operations:

pop()

Example

Storage:

5

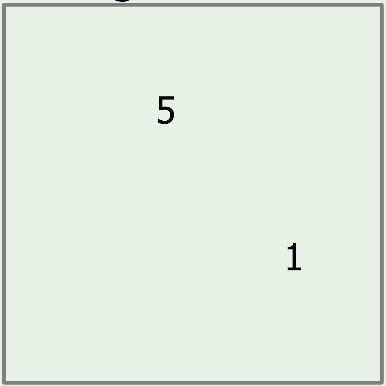
1

Operations:

pop() → 7

Example

Storage:

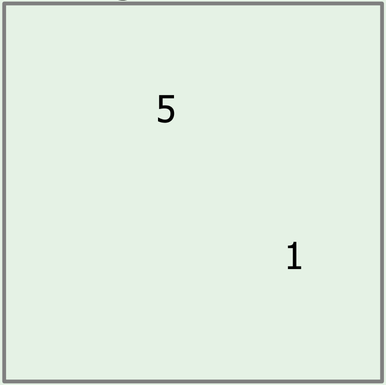


Operations:

top()

Example

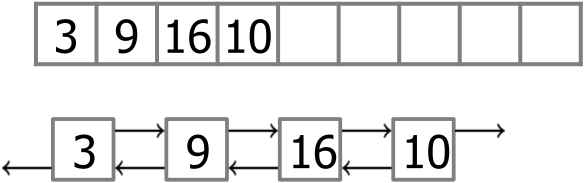
Storage:



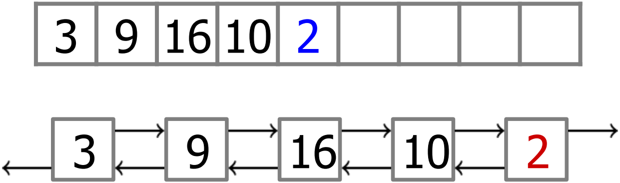
Operations:

top() → 5

Implementing Priority Queue with Unsorted Array/List



Implementing Priority Queue with Unsorted Array/List

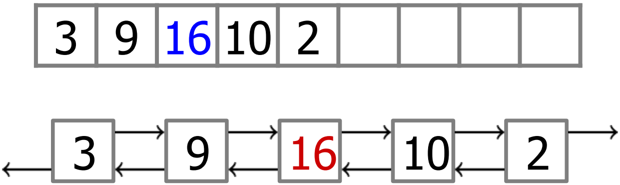


insert(e)

add *e* to the end

running time: $O(1)$

Implementing Priority Queue with Unsorted Array/List



insert(e)
add *e* to the end
running time: *O*(1)

pop()
scan array/list to find max
running time: *O*(n)

Implementing Priority Queue with Sorted Array

2	3	9	10	16				

pop()

extract the last element

running time: $O(1)$

Implementing Priority Queue with Sorted Array

2	3	9	10	16				

pop()

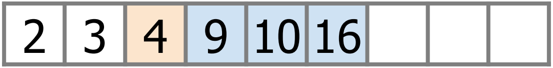
extract the last element

running time: $O(1)$

insert(e)

find a position for e using binary search: $O(\log n)$

Implementing Priority Queue with Sorted Array



pop()

extract the last element

running time: $O(1)$

insert(e)

find a position for e using binary search: $O(\log n)$

shift all elements to the right of it by 1: $O(n)$

insert e: $O(1)$

running time: $O(n)$

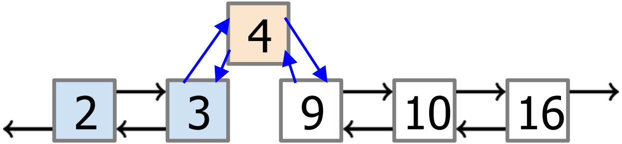
Implementing Priority Queue with Sorted List

pop()

extract the last element

running time: $O(1)$

Implementing Priority Queue with Sorted List



pop()

extract the last element

running time: $O(1)$

insert(e)

find a position for *e* (cannot use binary search): $O(n)$

insert *e*: $O(1)$

running time: $O(n)$

Priority Queue: running time for different implementations

	insert	pop
Unsorted array/list	O(1)	O(n)
Sorted array/list	O(n)	O(1)

Many algorithms use Priority Queues

- Dijkstra’s algorithm: finding a shortest path in a graph
- Prim’s algorithm: constructing a minimum spanning tree of a graph
- Huffman encoding: constructing an optimum prefix-free encoding of a string
- Heap sort: sorting a given sequence

Priority Queue: running time for different implementations

	insert	pop
Unsorted array/list	O(1)	O(n)
Sorted array/list	O(n)	O(1)
Binary Heap	O(log n)	O(log n)