# Algorithms and Data Structures

## Queues: Searching Lists (revisited)

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

# Outline

## Warm Up

Www.menti.com, check.

Hand-in Assignments, check.

## Queues

Queues, Check-it

LIFO - Stacks

FIFO

Priority Queues

## Hand-in Assignment #3

Content, scope

## Warm Up

Www.menti.com, check.

Hand-in Assignments, check.

## Queues

Queues, Check-it

LIFO - Stacks

FIFO

Priority Queues

## Hand-in Assignment #3

Content, scope

cphbusiness
COPENHAGEN BUSINESS ACADEMY

**A quick test on complexity**

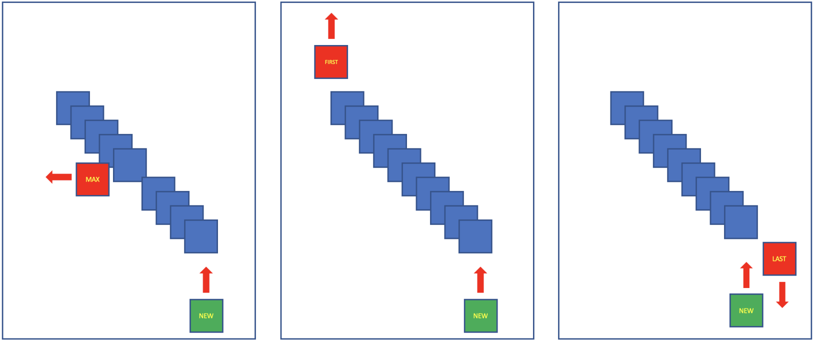
- Go to www.menti.com, and participate. Try to sub mit your answer *before* looking at the whiteboard.
- How did we do?

  *Should we do a mid-term review at some point?*

**Hand-in assignment #2 (Searching Shakespeare)**

- Handed in? - Yes? - No?
- Conducted peer review on Hand-in #1? Yes? No?
- Are the assignments doable?

Let's take a look at it; what can we do in queueing, generally speaking.



# LIFO Queues - Stacks

Interface of LIFO Queues - Stacks

```
interface Stack<T> {
  void push(T item);
  T pop() throws NoSuchElementException;
  T peek() throws NoSuchElementException;
  int size();
  default boolean isEmpty() { return size() = 0; }
  }
```

Interface of FIFO Queues

```java
interface Queue<T> {
  void enqueue(T item);
  T dequeue() throws NoSuchElementException;
  T peek() throws NoSuchElementException;
  int size();
  default boolean isEmpty() { return size() = 0; }
  }
```

# Exercise 1 - Array queue

Implement the `Queue` inferface using an **array** data structure.

```
interface Queue<T> {
  void enqueue(T item);
  T dequeue() throws NoSuchElementException;
  T peek() throws NoSuchElementException;
  int size();
  default boolean isEmpty() { return size() = 0; }
  }
```

Implement the `Queue` inferface using a **linked list** data structure.

```
interface Queue<T> {
  void enqueue(T item);
  T dequeue() throws NoSuchElementException;
  T peek() throws NoSuchElementException;
  int size();
  default boolean isEmpty() { return size() = 0; }
  }
```

Interface of Priority Queues

```
interface PriorityQueue<T extends Comparable<T>> {
  void enqueue(T item);
  T dequeue() throws NoSuchElementException;
  T peek() throws NoSuchElementException;
  int size();
  default boolean isEmpty() { return size() = 0; }
  }
```

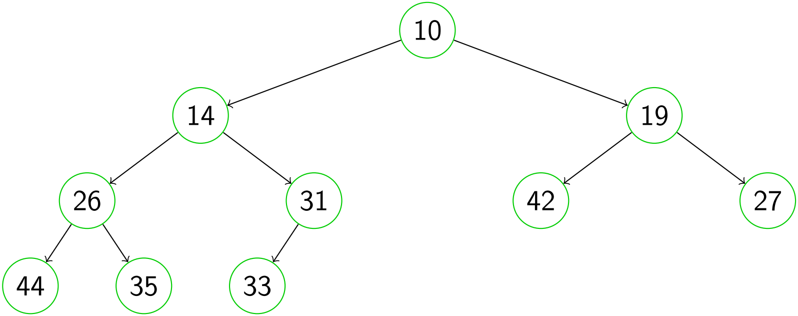
# Priority Queue Implementations

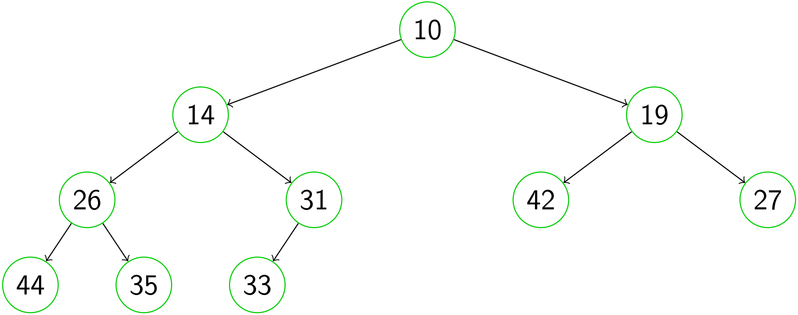
- Search for top item every time.
  - insert: $O(1)$
  - dequeue: $O(n)$
- Sort data structure, keep sorted at inserts
  - insert: $O(n)$
  - dequeue: $O(1)$
- Use a semisorted structure, a heap
  - insert: $O(\log n)$
  - dequeue: $O(\log n)$

Heaps are semisorted binary trees:

- The root of a heap holds the extreme element (maximum/minimum)
- The branches of a heap are:
  - Heaps themselves
  - Empty nodes
- Heaps are balanced
- Filled from “left” to “right”

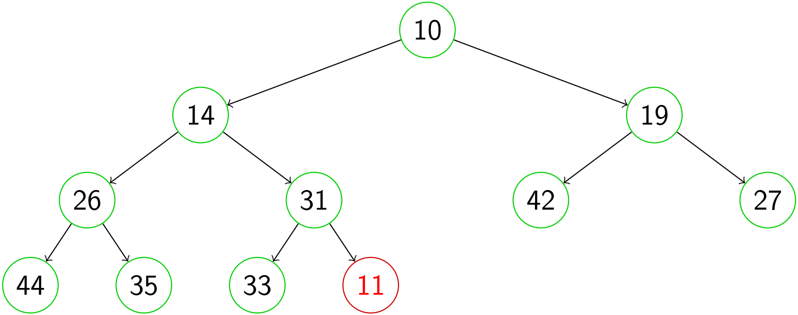
# Min-Heap





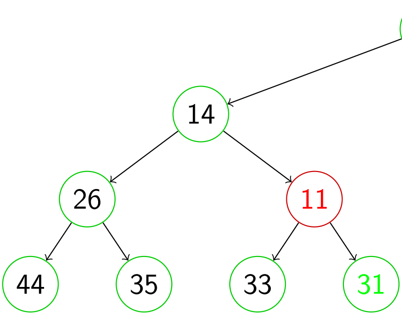

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 14 | 19 | 26 | 31 | 42 | 27 | 44 | 35 | 33 | | | | | |

enqueue




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 14 | 19 | 26 | 31 | 42 | 27 | 44 | 35 | 33 | 11 | | | | |

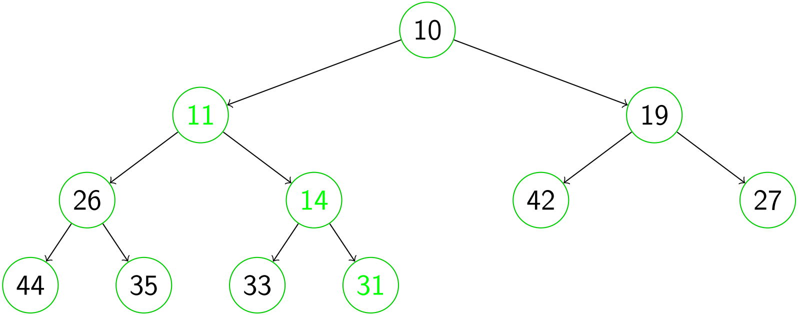
enqueue




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 14 | 19 | 26 | 11 | 42 | 27 | 44 | 35 | 33 | 31 | | | | |

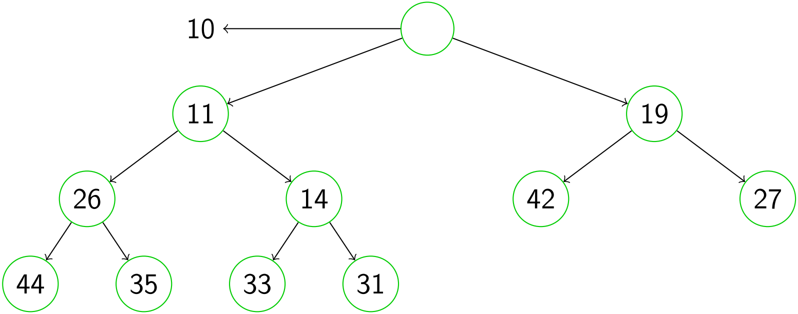
# Heap

enqueue



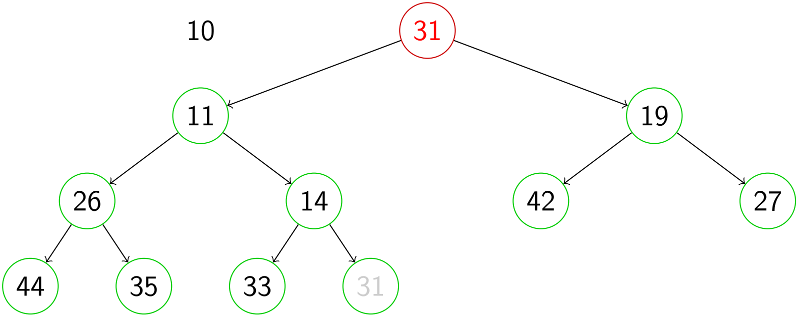

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 19 | 26 | 14 | 42 | 27 | 44 | 35 | 33 | 31 | | | | |

dequeue



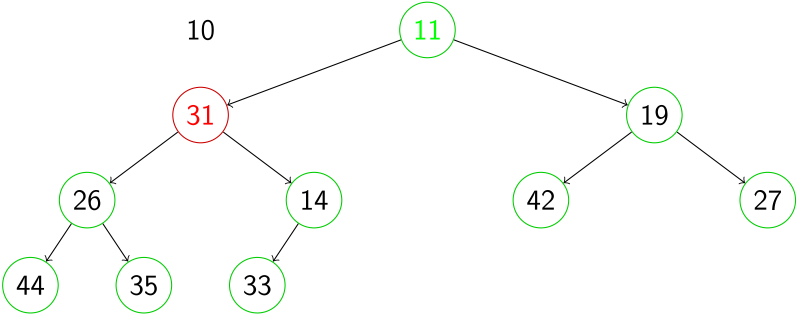

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11 | 19 | 26 | 14 | 42 | 27 | 44 | 35 | 33 | 31 | | | | |

dequeue



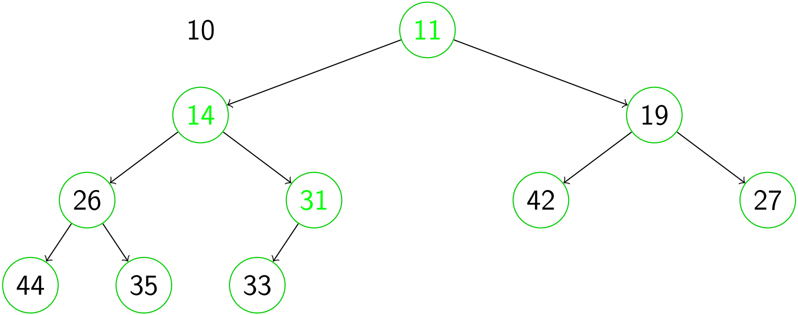

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 11 | 19 | 26 | 14 | 42 | 27 | 44 | 35 | 33 | 31 | | | | |

dequeue




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 31 | 19 | 26 | 14 | 42 | 27 | 44 | 35 | 33 | | | | | |

dequeue




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 14 | 19 | 26 | 31 | 42 | 27 | 44 | 35 | 33 | | | | | |

## Warm Up

- Www.menti.com, check.
- Hand-in Assignments, check.

## Queues

- Queues, Check-it
- LIFO - Stacks
- FIFO
- Priority Queues

## Hand-in Assignment #3

- Content, scope

## Airport Prioritized Queue

- Will build on queues (this week, and next).
- Make sure to read through the material in the book (pdf).
- You *may* (but must not) use the template provided in the Week 09 folder on GitHub.

In groups:

Implement a priotitized queueing system for an airport. You can use any priority queue algorithm, but you must be able to argue that the time complexity is no worse than $O(\log n)$ for enqueue and dequeue respectively.

You should implement the priority queue in a setup that simulates passengers arriving to an airport, and passengers passing security.

Passenger priority can be derived from the passenger category and arrival time:

1. Late to flight
2. Business class
3. Disabled
4. Family
5. Monkey