THINK BIG WE DO

CSC 212

Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Priority Queues & Heaps

Housekeeping

Review Project [MEC]

- Due October 28, 11:59pm

Election Day / Veteran’s Day

- Nov 7-11
- Class only meets Thursday, Nov 10
- Assignment 4 Due
- Lab 9: Balancing Act Due
  - In-person labs are canceled

PRIORITY QUEUES

Definitions

Collections

Insert and delete items. Which items to delete?

Stack

Remove the item most recently added

Randomized Queue

Remove a random item

Queue

Remove the item least recently added

Generalizes

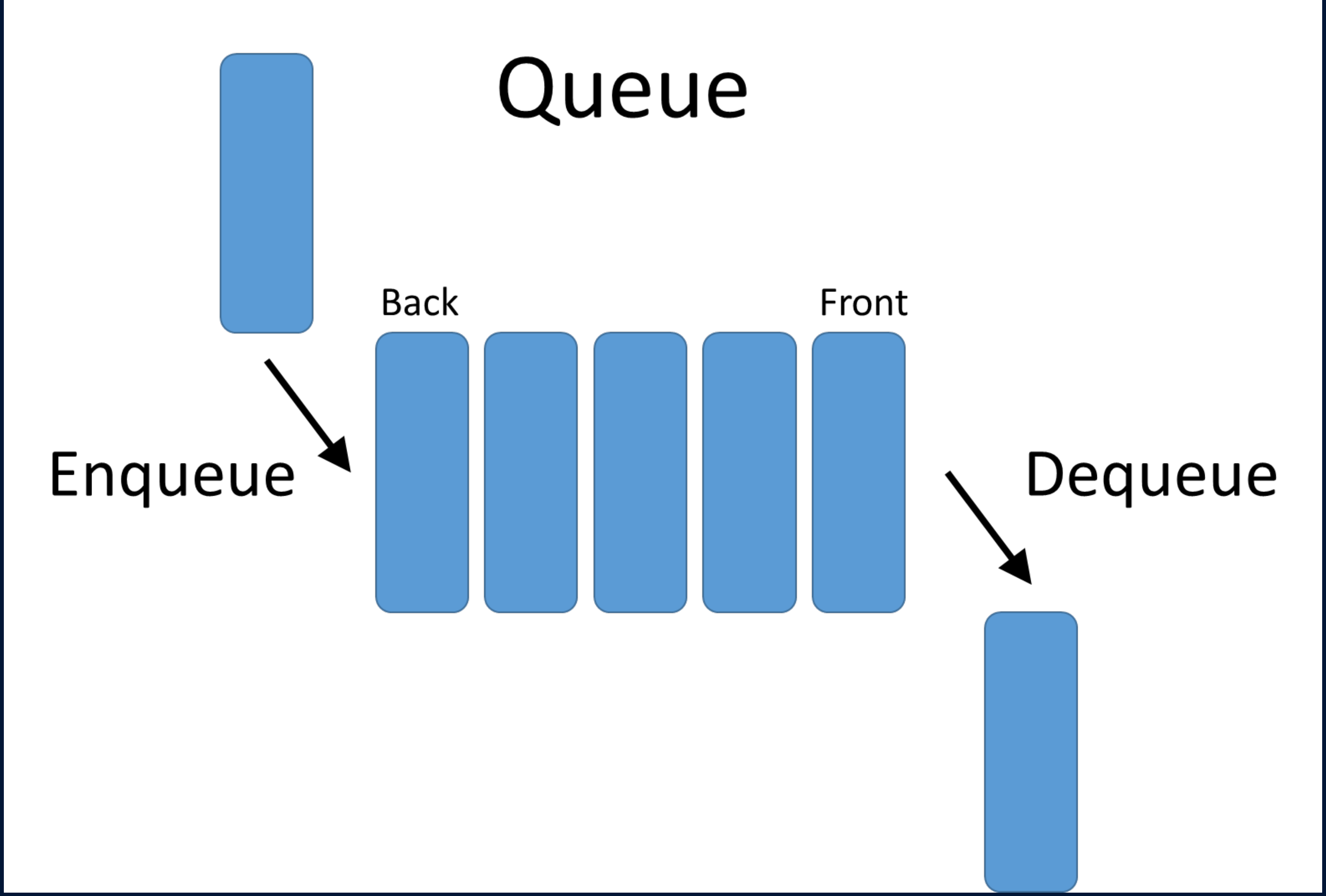
stack, queue, randomized queue

Priority Queue

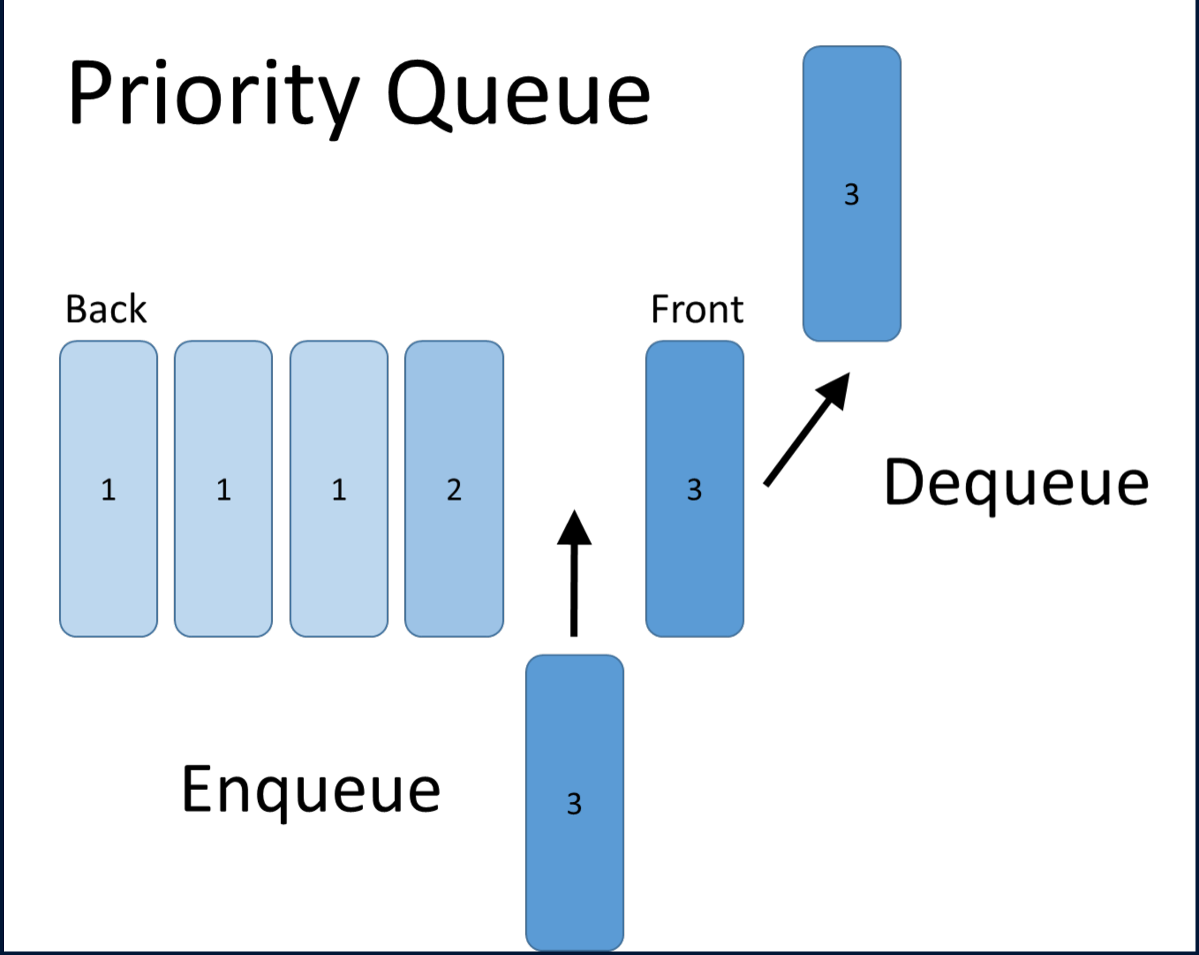
Remove the largest (or smallest) item

credit: COS 226, Princeton University

Queues



Priority Queue



Applications

Data Compression (huffman trees)

Network Routing

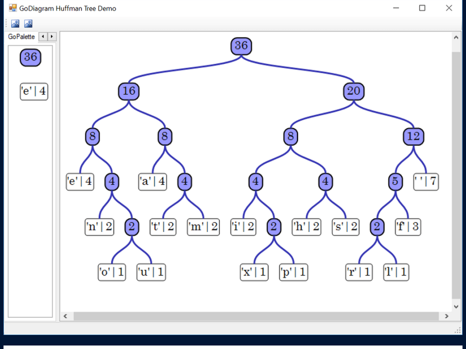
Process Schdeuling (CPUs)

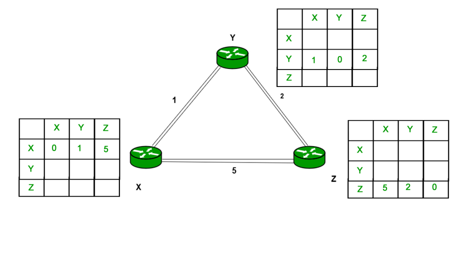
Artificial Intelligence (search)

Graph Algorithms

Stream Data Algorithms

HPC Task Scheduling

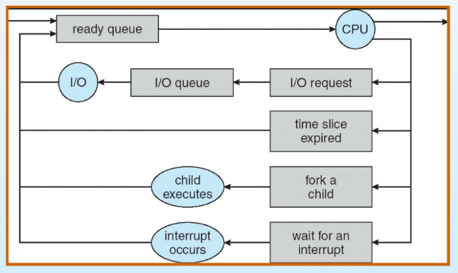




	X	Y	Z
X	0	1	5
Y			
Z			

	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0



ready queue

CPU

I/O

I/O queue

I/O request

time slice expired

child executes

fork a child

interrupt occurs

wait for an interrupt

Priority Queues

Collections of $< key, value >$ pairs

- *keys* are objects on which an *order* is defined

Every pair of keys must be comparable according to a *total order*:

Properties

Reflexive:	Antisymmetric:	Transitive:
$k_1 \leq k_2$	$k_1 \leq \; k_2 \quad \wedge \quad k_2 \leq k_1$	$k_1 \leq k_2 \quad \wedge \quad k_2 \leq k_3$
	$\Rightarrow$	$\Rightarrow$
	$k_1 = k_2$	$k_1 \leq k_3$

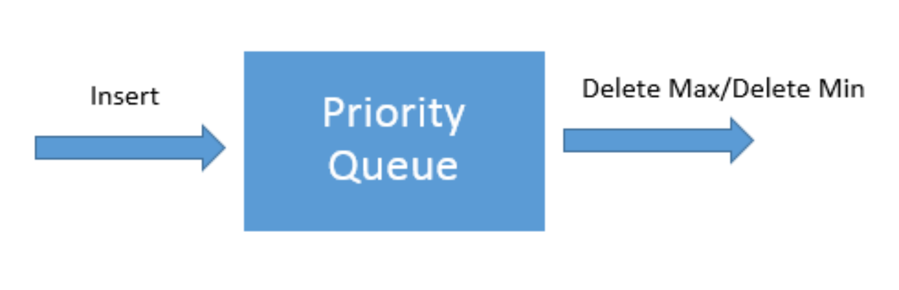
Priority Queues

Queues

- basic operations:
  - $enqueue, dequeue$
- always remove the item least recently added

Priority Queues

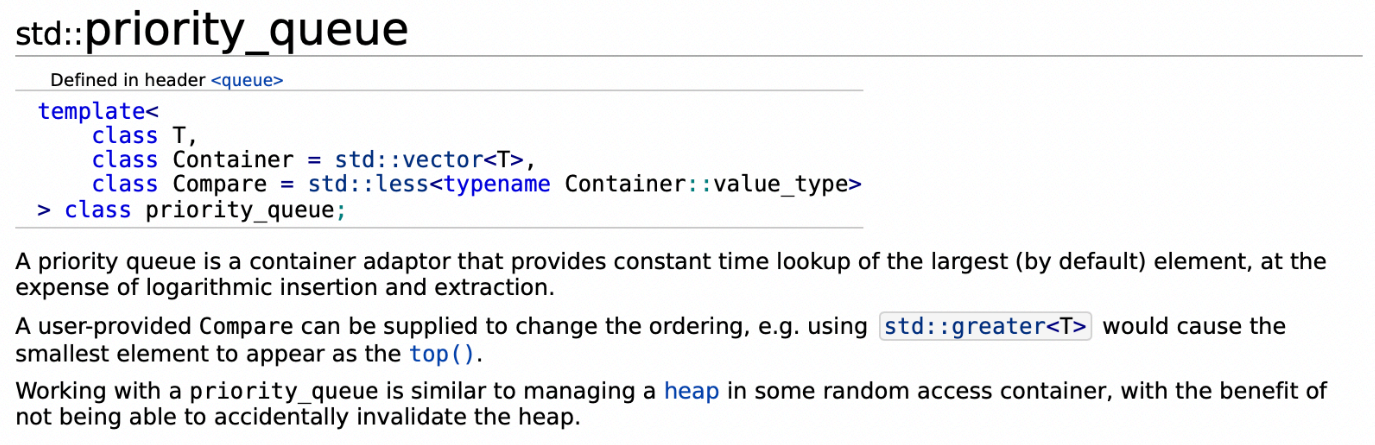
- basic operations:
  - $insert, removeMax$
- MaxPQ:
  - always remove the item with the highest (max) priority
- MinPQ:
  - always remove the item with the lowest (min) priority



Performance

	Sorted Array/List	Unsorted Array/List
insert	$O(n)$ must find place where to insert item	$O(1)$ item can be inserted at head or tail
removeMax max	$O(1)$ largest/smallest key is at: $arr[0]/arr[n-1]$	$O(n)$ must traverse entire sequence to find largest/smallest

cppreference.com



std::priority_queue

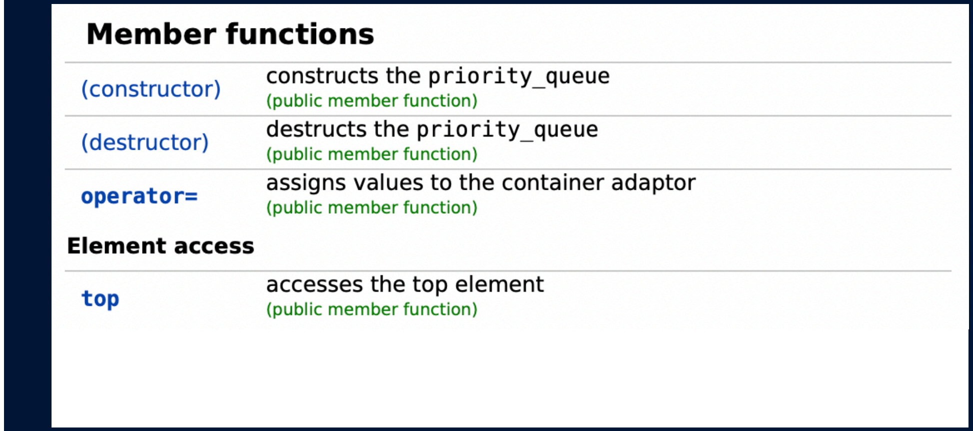
Defined in header <queue>

```
template<
    class T,
    class Container = std::vector<T>,
    class Compare = std::less<typename Container::value_type>
> class priority_queue;
```

A priority queue is a container adaptor that provides constant time lookup of the largest (by default) element, at the expense of logarithmic insertion and extraction.

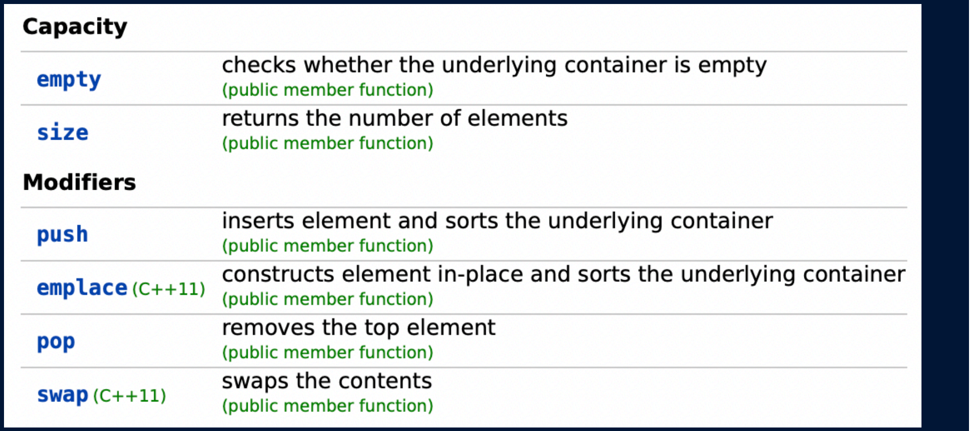
A user-provided `Compare` can be supplied to change the ordering, e.g. using `std::greater<T>` would cause the smallest element to appear as the `top()`.

Working with a `priority_queue` is similar to managing a heap in some random access container, with the benefit of not being able to accidentally invalidate the heap.



Member functions

(constructor)	constructs the `priority_queue` (public member function)
(destructor)	destructs the `priority_queue` (public member function)
operator=	assigns values to the container adaptor (public member function)
Element access	
top	accesses the top element (public member function)



Capacity	
empty	checks whether the underlying container is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element and sorts the underlying container (public member function)
emplace (C++11)	constructs element in-place and sorts the underlying container (public member function)
pop	removes the top element (public member function)
swap (C++11)	swaps the contents (public member function)

HEAPS

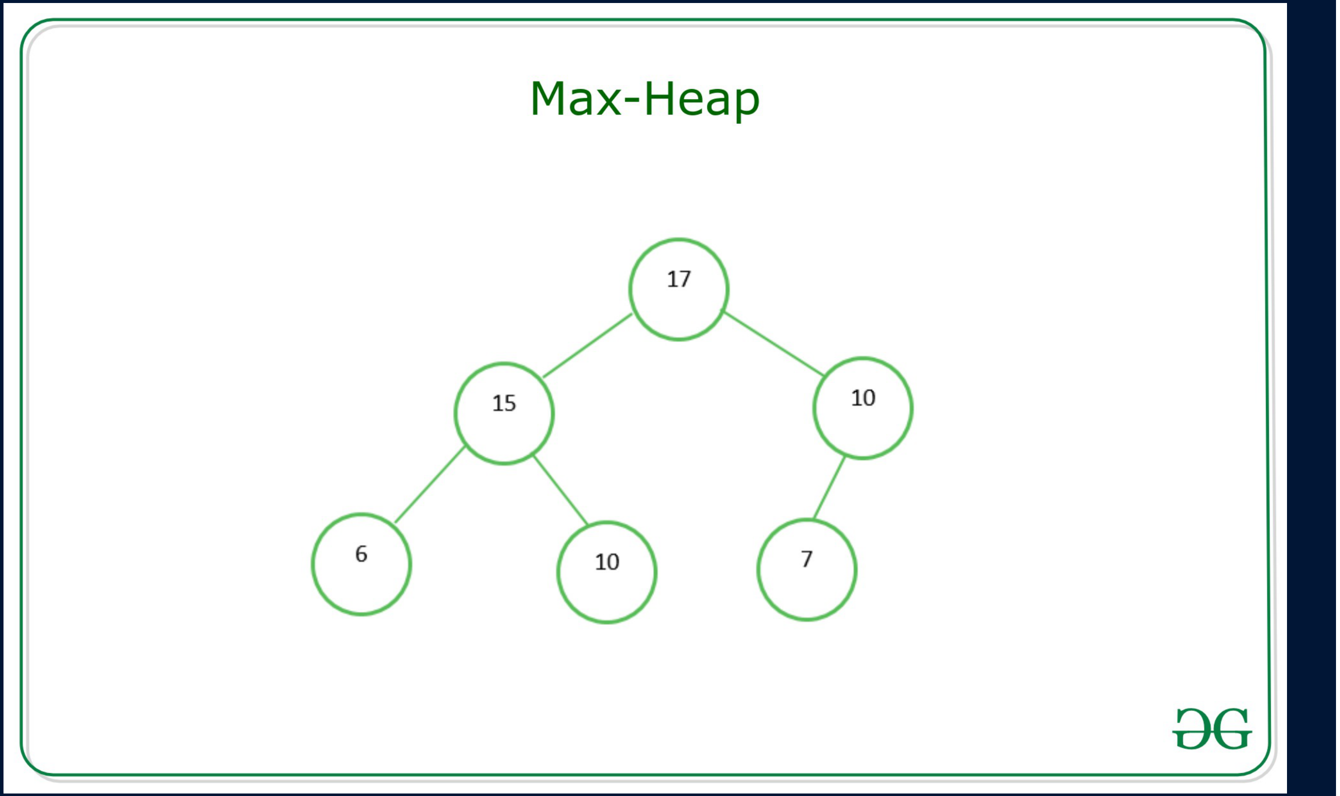
(max) Heap

Structure Property

- a heap is a *complete binary tree*

Heap-Order Property

- for every node x:
  - $key\ parent\ x \ \geq\ key\ x$
  - except the root, which has no parent



Height of a heap

What is the minimum number of nodes in a complete binary tree of height h?

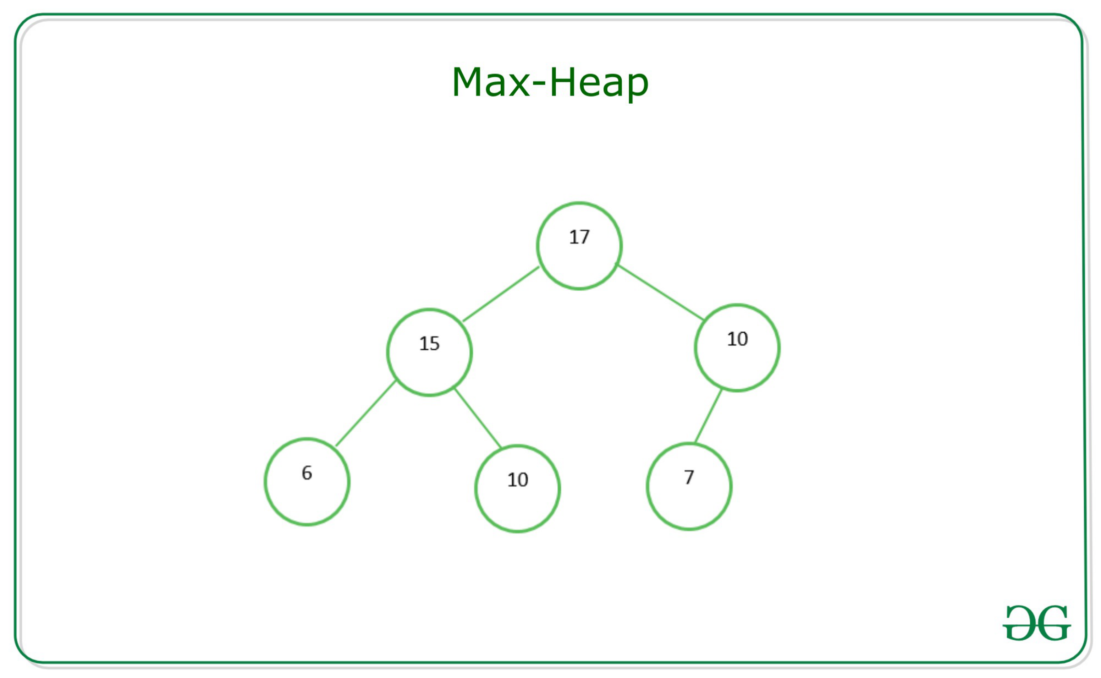
$$n \geq 2^h$$

$$\Rightarrow$$

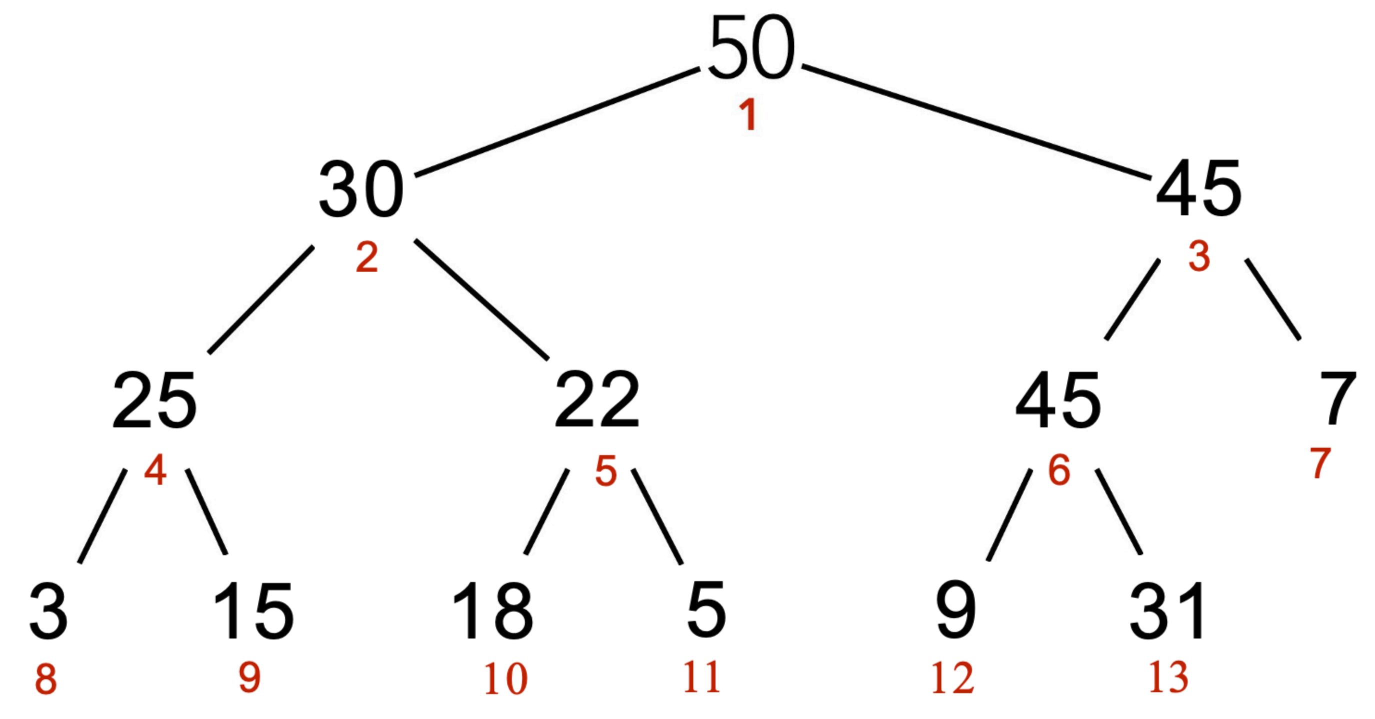
$$log\, n \geq log\, 2^h$$

$$\Rightarrow$$

$$log\, n \geq h$$



Implementation



node(i)

i

parent(i)

$floor(\frac{i}{2})$

left_child(i)

$i * 2$

right_child(i)

$i * 2 + 1$

Complete tree…

50	30	45	25	22	45	7	3	15	18	5	9	31
1	2	3	4	5	6	7	8	9	10	11	12	13

$insert$

Append new element to the end of array

Check heap-order property

- if violated, *Up-Heap* (swap with parent)
    - **repeat** until heap-order is restored
- if not, $insert$ complete

Time Complexity

- $O(log\,n)$

insert

$removeMax$

Max element is the first element of the array

- the $root$ of the heap

Copy last element of array to the first position

- then decrement array size by 1 (removes the last element)

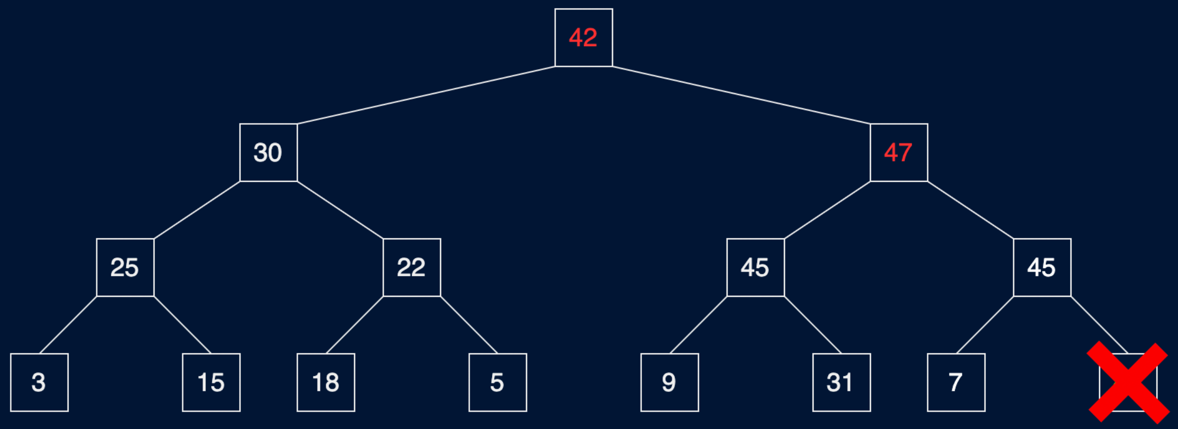
Check heap-order property

- if violated, *Down-Heap* (swap with **larger** child)
  - **repeat** until heap-order is restored
- if not, *insert* complete

Time Complexity

- $O(log\, n)$

$removeMax$



42	30	45	25	22	45	7	3	15	18	5	9	31	47		

- tree ⇒ move 42 to root
- validate heap-order…

Performance

	Sorted Array/List	Unsorted Array/List		Heap
insert	$O(n)$	$O(1)$		$O(log\, n)$
removeMax	$O(1)$	$O(n)$		$O(log\, n)$
max	$O(1)$	$O(n)$		$O(1)$
insert N	$O(n^2)$	$O(n)$		$O(n)^{**}$

(**) assuming we know the sequence in advance (*buildHeap*)