CSCI 136: Data Structures and Advanced Programming

Lecture 25

Trees, part 5

Instructor: Dan Barowy

Williams

Announcements

Office hours today: 5-7pm

1st years: academic advising.

Pre-registration info session: 4-5pm, Wege Auditorium

Speaker: Steve Lombardi from Oculus, 2:30-4pm, Wege Auditorium

Outline

Review: Priority queues

Heaps

Quiz

Recall: Priority Queues

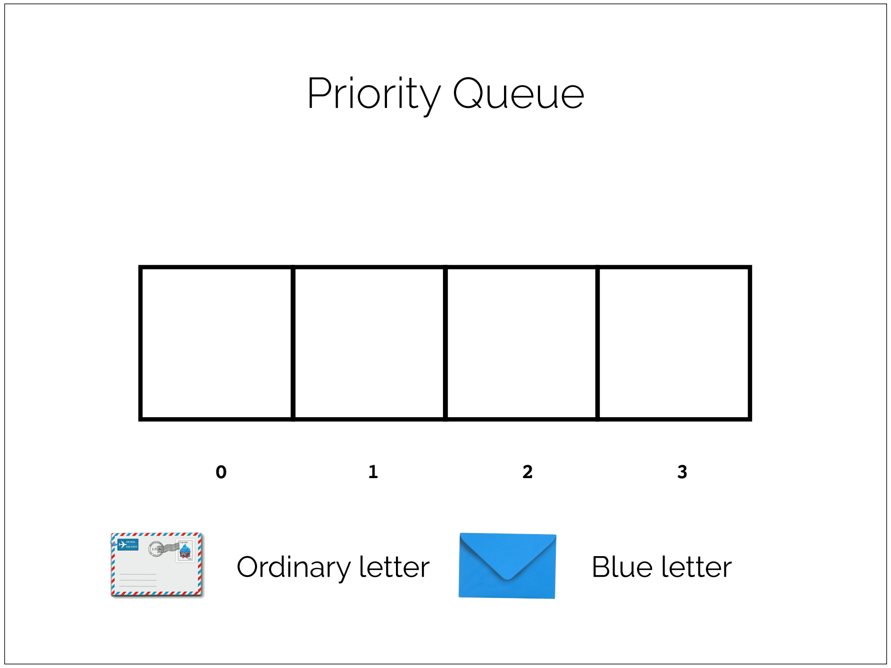
Priority Queue

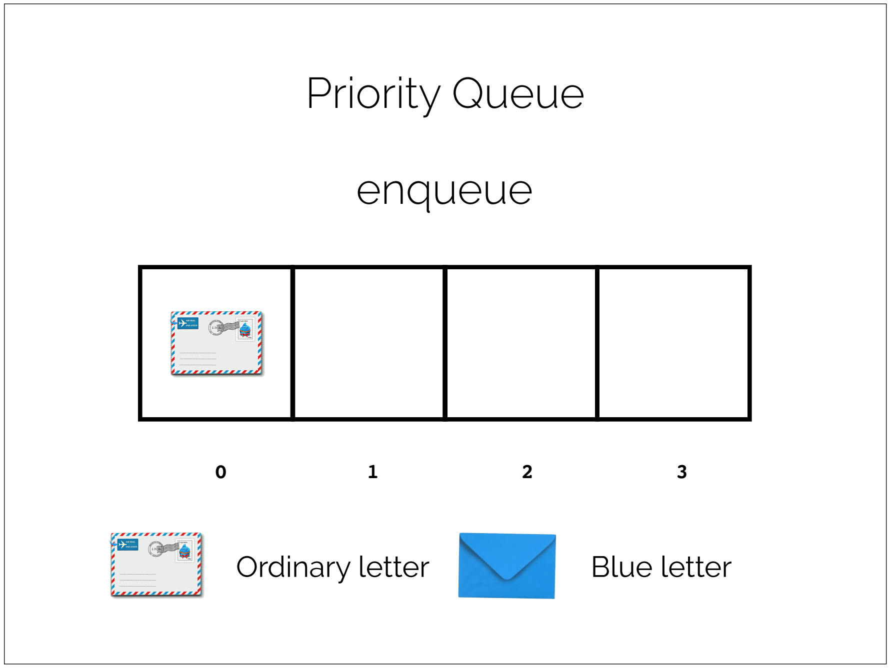
A **priority queue** is an abstract data type that returns the elements in **priority order**. Under priority ordering, an element **e** with a higher priority (an integer) is returned before all elements **L** having lower priority, even if that **e** was enqueued after all **L**. When any two elements have **equal priority**, they are returned in **first-in, first-out order** (i.e., in the order in which they were enqueued).

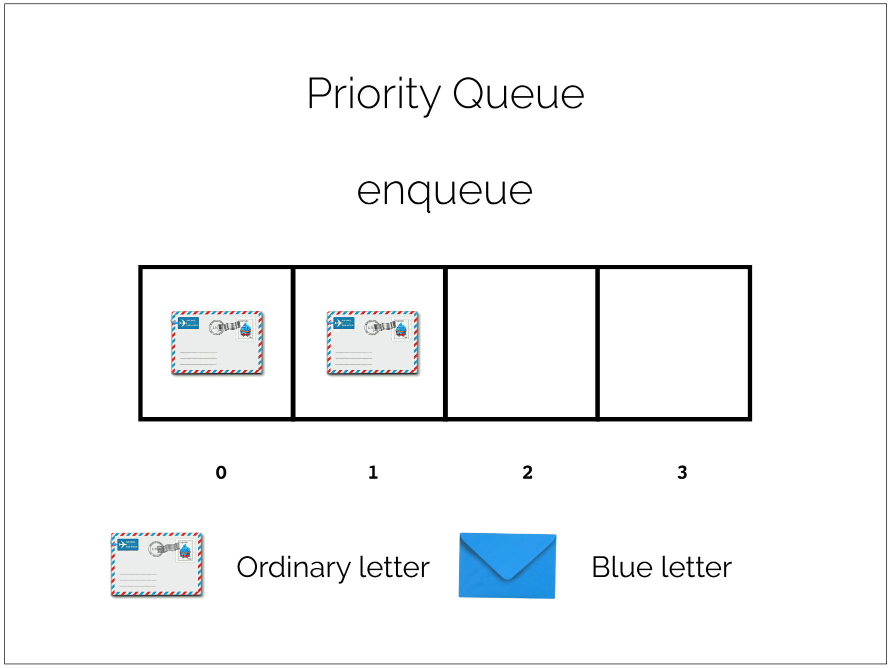
Note

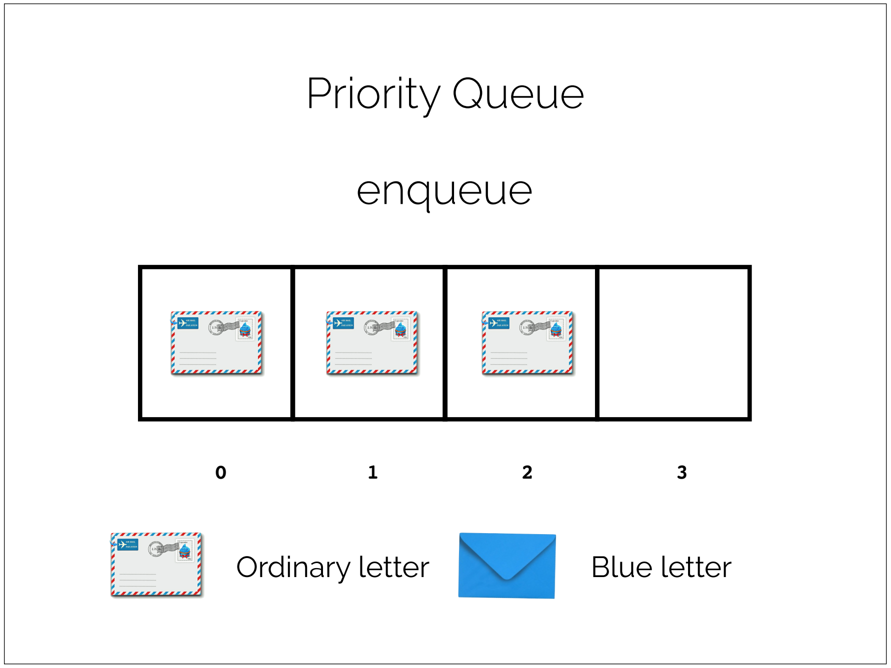
I will refer here to the **maximum** priority. But you could also refer to **minimum** priority. All that matters is that you order your data with respect to some **extremum**.

Blue letter


Priority Queue
0
1
2
3
Ordinary letter
Blue letter

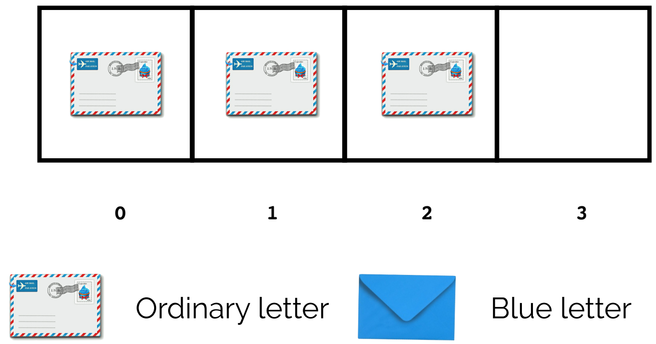

Priority Queue
enqueue
0
1
2
3
Ordinary letter
Blue letter


Priority Queue
enqueue
0
1
2
3
Ordinary letter
Blue letter


Priority Queue
enqueue
0
1
2
3
Ordinary letter
Blue letter

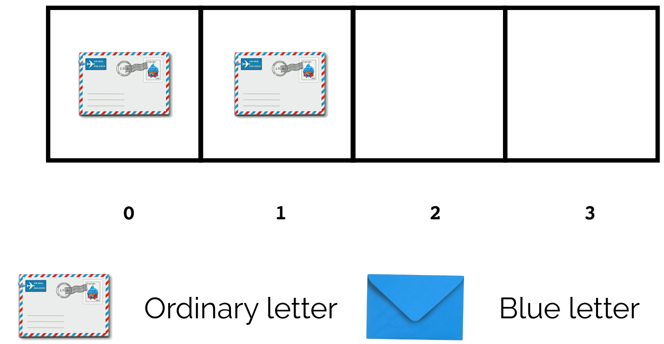
Priority Queue

extract



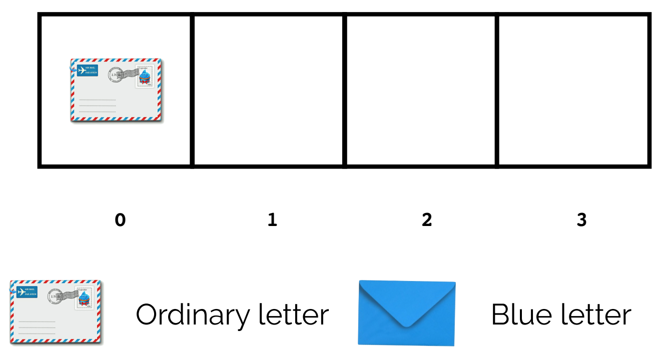
Priority Queue

extract



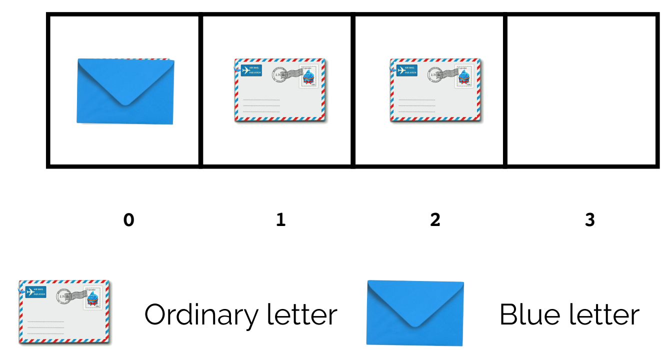
Priority Queue

extract



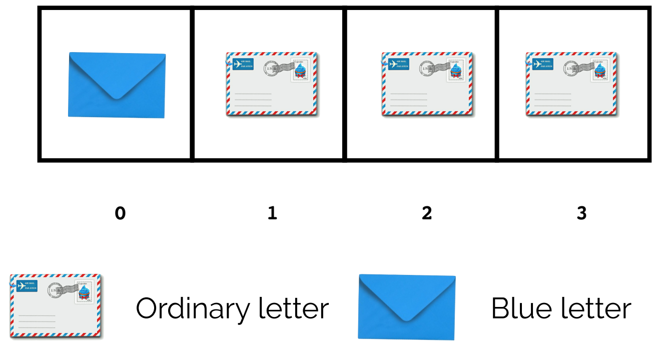
Priority Queue

blue letters: enqueue



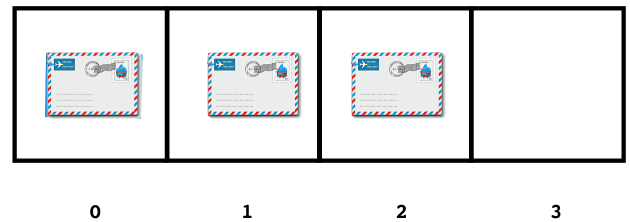
Priority Queue

blue letters: extract



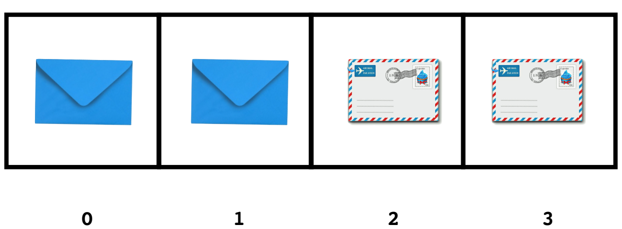
Priority Queue: Operations

insert: inserts an element with a given priority value. Ensures that the next element of the queue is in priority order. Like **enqueue**.



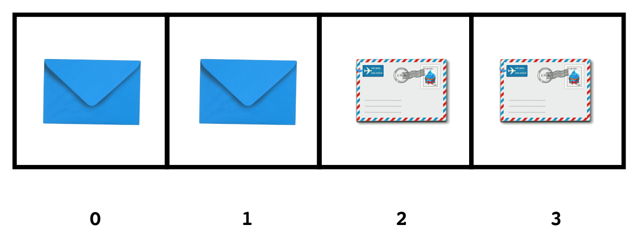
Priority Queue: Operations

find-max: returns the next element with a highest priority value. Like **peek**, does not modify the queue.



Priority Queue: Operations

extract: removes and returns the next element with a maximum priority value. Like **dequeue**.



Priority Queue

How to implement?

Vector:
- **find-max**: O(1)
- **insert**: O(n)
- **extract**: O(n)

BinarySearchTree:
- **find-max**: O(n)
- **insert**: O(n)
- **extract**: O(n)

Heap:
- **find-max**: O(1)
- **insert**: O(log n)
- **extract**: O(log n)

Priority Queue

Is it **necessary** to keep the **entire queue** in sorted order?

Operations:

find-max
insert
extract

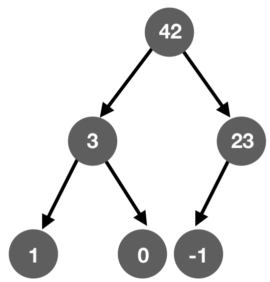
Heaps

Max Heap

A **max heap** is a tree-based data structure that returns its elements in **priority order**. A heap maintains the **max heap property**: for any given node **n**, if **p** is a parent node of **n**, then the **key** of **p** is ≥ to the **key** of **n**.

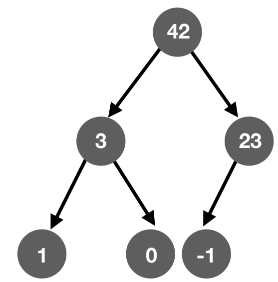
A max heap is a tree whose root is the maximum element and whose subtrees are, themselves, heaps.

Is this a binary search tree?



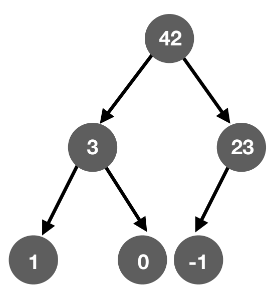
No. Nodes do not obey **binary search property**.

(Binary) max heap



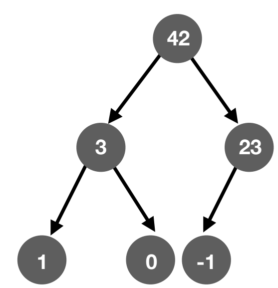
Max heap property: for any given node **n**, if **p** is a parent node of **n**, then the **key** of **p** is ≥ the **key** of **n**.

Insertion



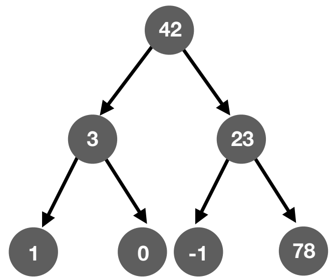
A **binary heap** is usually implemented as an **always-complete binary tree**.

Insertion



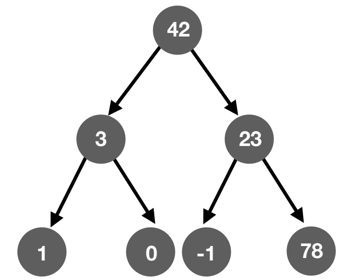
Suppose we want to insert a new node, 78

Insertion



First, **insert** the new node at the first available position in the tree that **maintains completeness**.

Insertion

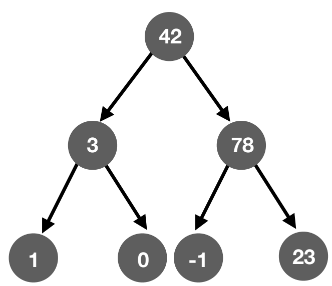


23 ≥ 78 ?

No.

Next, **compare** the new node with its parent.

Insertion

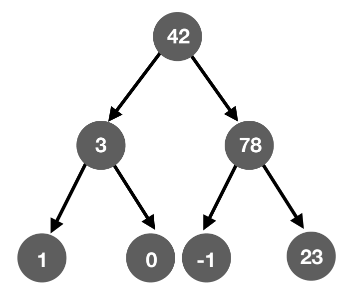


23 ≥ 78 ?

No.

If the **max heap property** is violated, **swap**.

Insertion

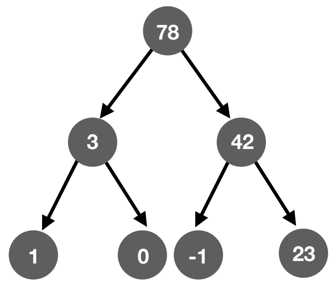


42 ≥ 78 ?

No.

Continue swapping the new node with parents until the **max heap property is satisfied**.

Insertion

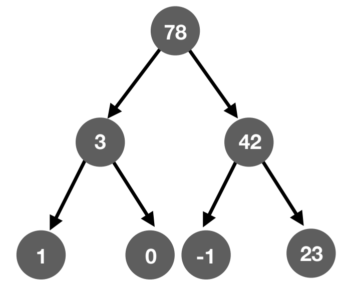


42 ≥ 78 ?

No.

Continue swapping the new node with parents until the **max heap property is satisfied** (parent ≥ node or no parents remain).

Insertion

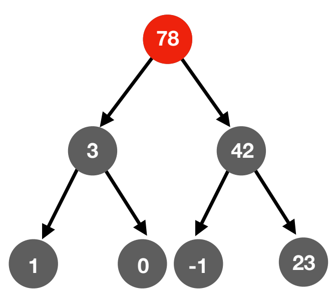


42 ≥ 78 ?

No.

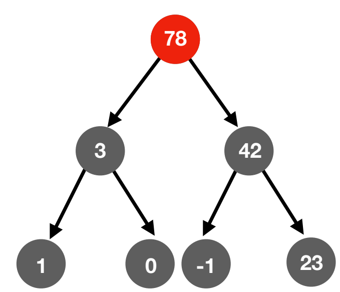
The **swapping procedure** performed on **insert** is often referred to as **heap-up** or **percolate-up**.

Find-max



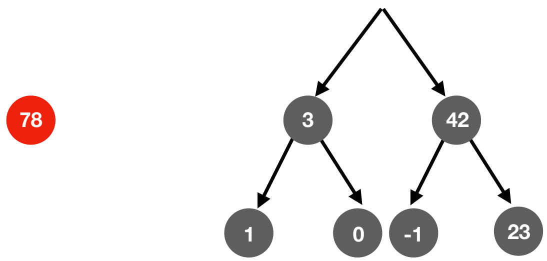
To find the **maximum element** in a max heap, simply **return** the **root**.

Extract



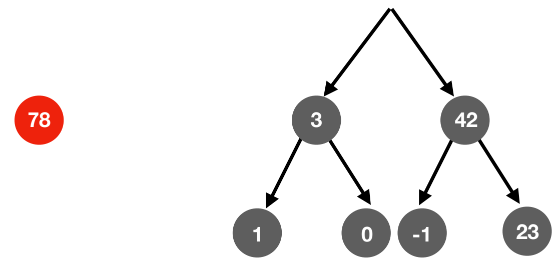
To **remove and return** the **maximum element** in a max heap, first perform **find-max**.

Extract



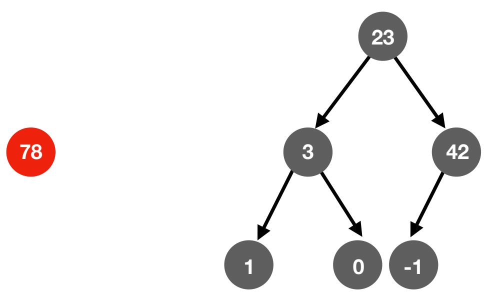
Temporarily store the max element.

Extract



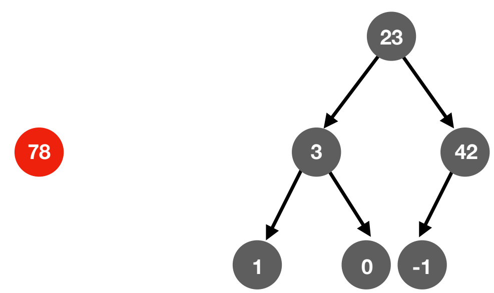
Replace the **root** with the **last element** in the complete tree.

Extract



Replace the **root** with the **last element** in the complete tree.

Extract

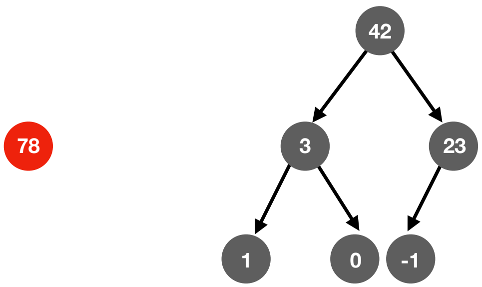


$23 \geq 42$?

No.

Compare the root with its children. **Swap** the **root** with **the largest element**.

Extract

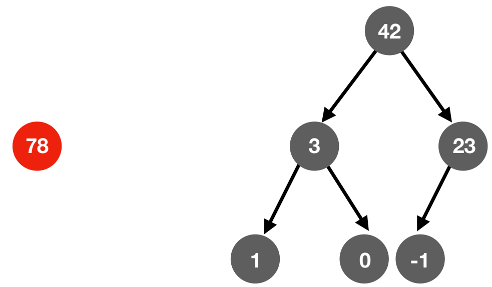


23 ≥ 42 ?

No.

Compare the root with its children. **Swap** the **root** with **the largest element**.

Extract

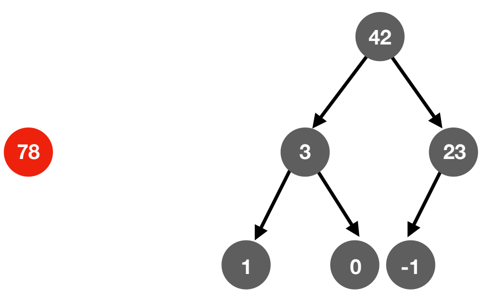


23 ≥ -1 ?

Yes.

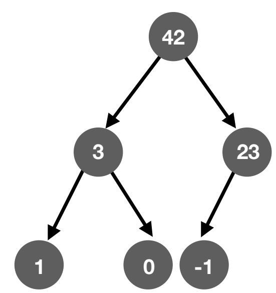
Continue swapping until the **max heap property is satisfied** (parent ≥ node or no parents remain).

Extract



Return the saved maximum element.

Extract

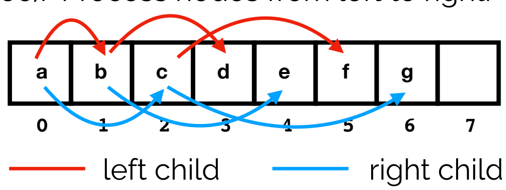


The **swapping procedure** performed on **extract** is often referred to as **heap-down** or **percolate-down**.

Activity

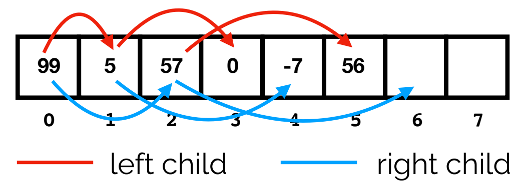
Build a max heap from the following elements:

But store the elements in an array (i.e., an implicit binary tree). Process nodes from left to right.




```
leftChild(i) = 2 × i + 1
rightChild(i) = 2 × i + 2
parent(i) = ⌊(i − 1) / 2)⌋
```

Implementation



A binary heap is often implemented using an implicit binary tree data structure. In other words, heap nodes are actually stored in an array or vector.

Advantages:

find-max: O(1)

insert: O(log n)

extract: O(log n)

Lots of interesting variants on heaps!

Summary of running times [edit]

In the following time complexities[5] *O*(*f*) is an asymptotic upper bound and *Θ*(*f*) is an asymptotically tight bound (see Big O notation). Function names assume a min-heap.

Operation	find-min	delete-min	insert	decrease-key	merge
Binary[5]	*Θ*(1)	*Θ*(log *n*)	*O*(log *n*)	*O*(log *n*)	*Θ*(*n*)
Leftist	*Θ*(1)	*Θ*(log *n*)	*Θ*(log *n*)	*O*(log *n*)	*Θ*(log *n*)
Binomial[5]	*Θ*(log *n*)	*Θ*(log *n*)	*Θ*(1)[a]	*Θ*(log *n*)	*O*(log *n*)[b]
Fibonacci[5][6]	*Θ*(1)	*O*(log *n*)[a]	*Θ*(1)	*Θ*(1)[a]	*Θ*(1)
Pairing[7]	*Θ*(1)	*O*(log *n*)[a]	*Θ*(1)	*o*(log *n*)[a][c]	*Θ*(1)
Brodal[10][d]	*Θ*(1)	*O*(log *n*)	*Θ*(1)	*Θ*(1)	*Θ*(1)
Rank-pairing[12]	*Θ*(1)	*O*(log *n*)[a]	*Θ*(1)	*Θ*(1)[a]	*Θ*(1)
Strict Fibonacci[13]	*Θ*(1)	*O*(log *n*)	*Θ*(1)	*Θ*(1)	*Θ*(1)
2-3 heap	?	*O*(log *n*)[a]	*O*(log *n*)[a]	*Θ*(1)	?

a. ^ a b c d e f g h i Amortized time.

b. ^ *n* is the size of the larger heap.

c. ^ Lower bound of $\Omega(\log\log n)$,[8] upper bound of $O(2^{2\sqrt{\log\log n}})$.[9]

d. ^ Brodal and Okasaki later describe a persistent variant with the same bounds except for decrease-key, which is not supported. Heaps with *n* elements can be constructed bottom-up in *O*(*n*).[11]

From **Wikipedia**: **priority queue** page.

Recap & Next Class

Today we learned:

Priority queues

Heaps

Next class:

Graphs