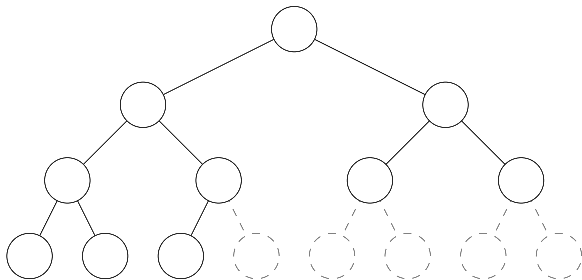
Heaps and Priority Queues

Heaps

Complete Binary Tree

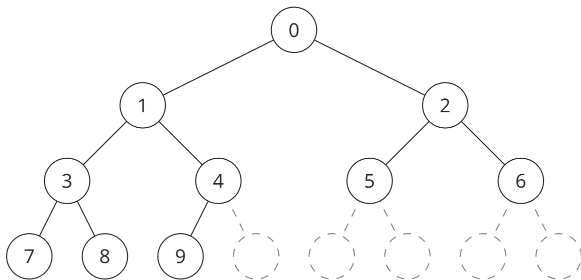
Complete binary tree

- All layers (except lowest) are full.
- Lowest layer is filled from left to right.



Complete Binary Tree – Array Representation

	1-Based Array	0-Based Array
Parent	$\left\lfloor \frac{i}{2} \right\rfloor$	$\left\lfloor \frac{i-1}{2} \right\rfloor$
Left child	$2i$	$2i+1$
Right child	$2i+1$	$2i+2$



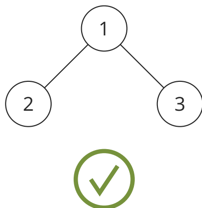
(Min) Heap Property

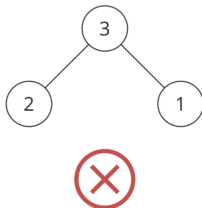
Min Heap Property

A complete binary tree satisfies *min heap property* if, for each node i which is not the root,

$$\text{key}(\text{parent}(i)) \leq \text{key}(i).$$

In case of an array A: $A[\text{parent}(i)] \leq A[i]$.





Heap

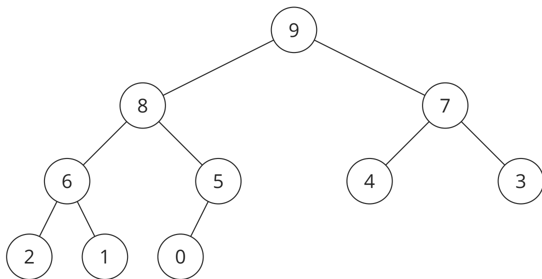
Heap

A *heap* is an array based representation of a complete binary tree satisfying the heap property.

Building a Heap

Building a Heap

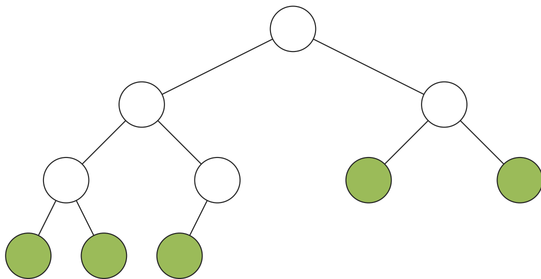
We have given an unsorted array. How do we transform it into a heap?



Building a Heap

Observation

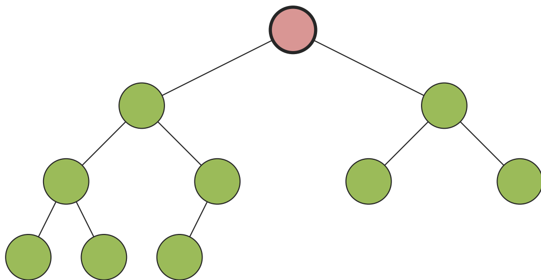
- Leafs are valid heaps.



Building a Heap

Idea

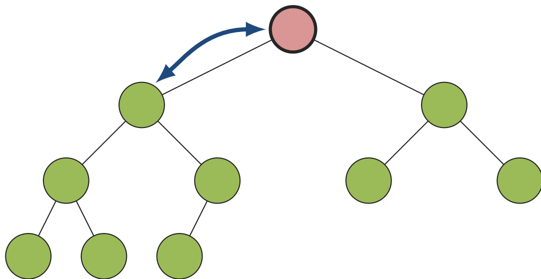
- Assume left and right subtrees are valid heaps.



Building a Heap

Idea

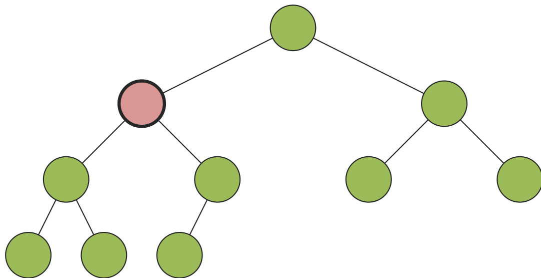
- Assume left and right subtrees are valid heaps.
- If necessary, exchange root with root of left or right subtree.



Building a Heap

Idea

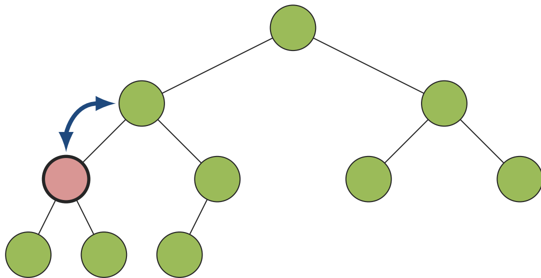
- Assume left and right subtrees are valid heaps.
- If necessary, exchange root with root of left or right subtree.
- Recursively repair subtree.



Building a Heap

Idea

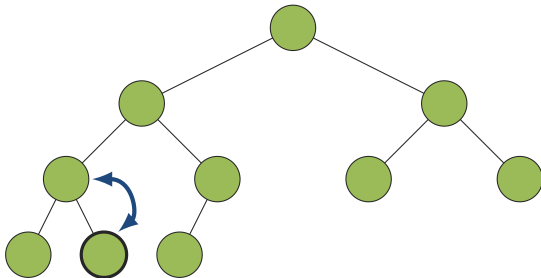
- Assume left and right subtrees are valid heaps.
- If necessary, exchange root with root of left or right subtree.
- Recursively repair subtree.



Building a Heap

Idea

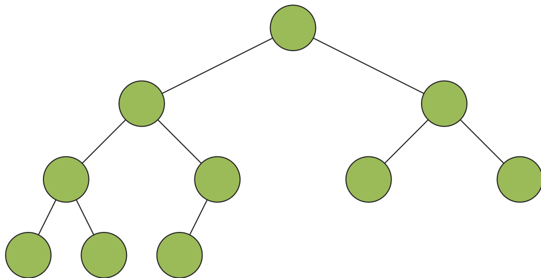
- Assume left and right subtrees are valid heaps.
- If necessary, exchange root with root of left or right subtree.
- Recursively repair subtree.



Building a Heap

Idea

- Assume left and right subtrees are valid heaps.
- If necessary, exchange root with root of left or right subtree.
- Recursively repair subtree.



Building a Heap — Algorithm

```
1 Procedure Min-Heapify (A, i)
2     Set l := left(i) and r := right(i).
3     If l ≥ |A| Then Return
4     If r < |A| and A[r] < A[l] Then smallest := r
5     Else smallest := l
6     If A[smallest] < A[i] Then
7         Exchange A[smallest] and A[i].
8         Min-Heapify(A, smallest)
```

The algorithm assumes 0-based arrays.

Building a Heap — Algorithm

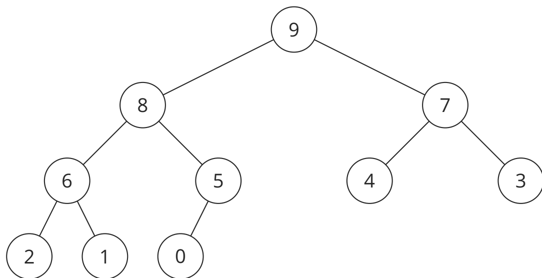
```
1 Procedure Build-Min-Heap (A)
2     For i := ⌊|A|/2⌋ − 1 DownTo 0
3         Min-Heapify(A, i)
```

The algorithm assumes 0-based arrays.

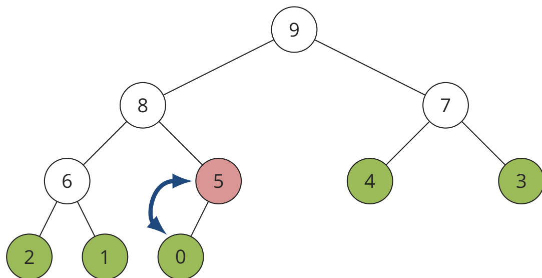
Theorem

The algorithm *Build-Min-Heap* runs in linear time.

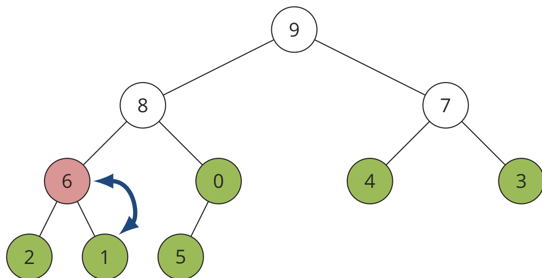
Example



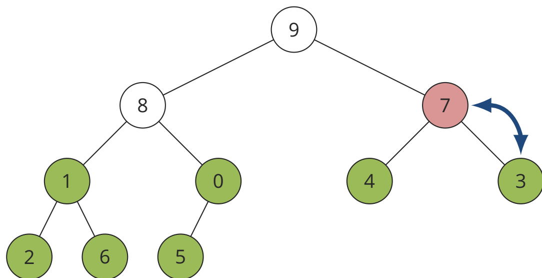
Example



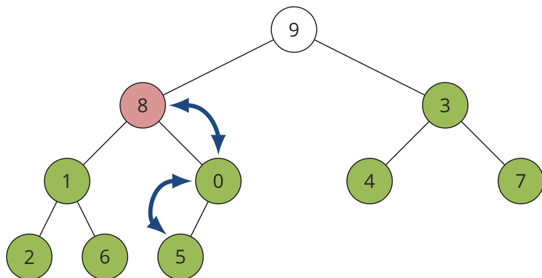
Example



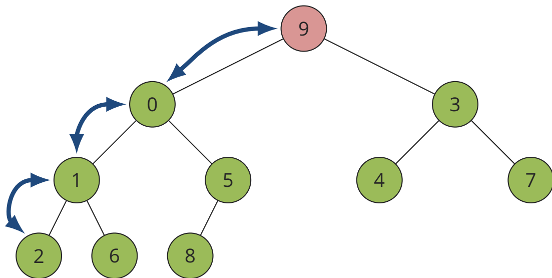
Example



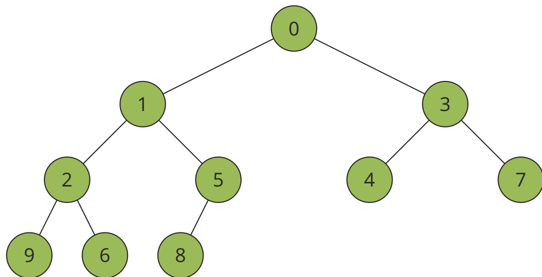
Example



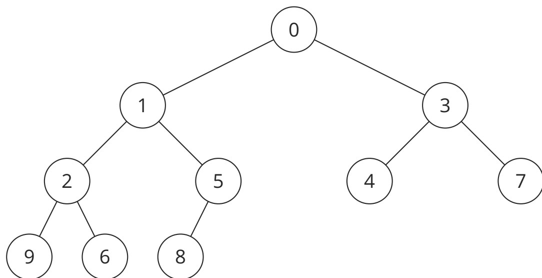
Example



Example



Example



Priority Queues

Queues (“FIFO")

Enqueue $\mathcal{O}(1)$

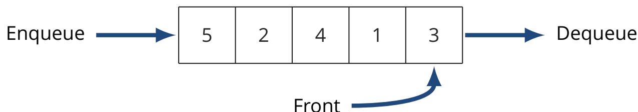
- Adds an element to the queue.

Dequeue $\mathcal{O}(1)$

- Removes the oldest element in the queue.

Front $\mathcal{O}(1)$

- Return the oldest element in the queue without removing it.



Priority Queues

Enqueue

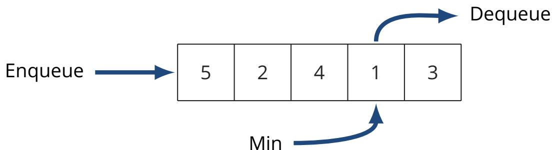
- Adds an element to the queue.

Dequeue

- Removes the ***smallest*** element in the queue.

Min

- Return the ***smallest*** element in the queue without removing it.

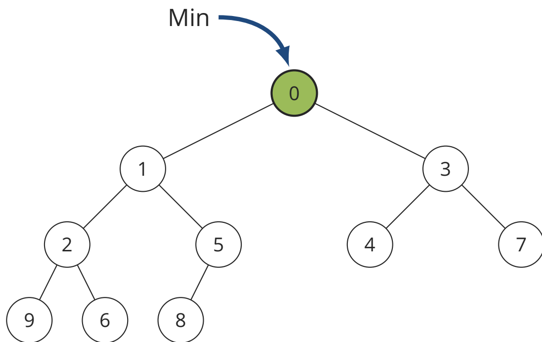


We can implement a priority queue using a heap.

Using a Heap – Min

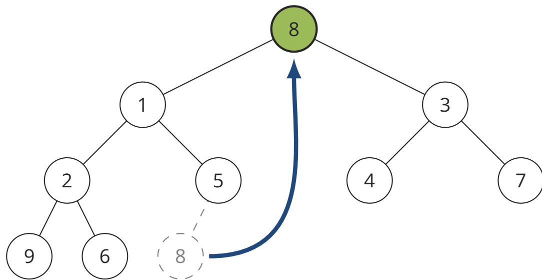
Finding the minimum

- The root of the heap
- $\mathcal{O}(1)$ time



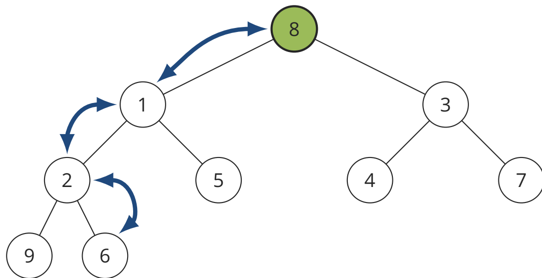
Dequeue

- “Remove” the root: Replace it by last element.



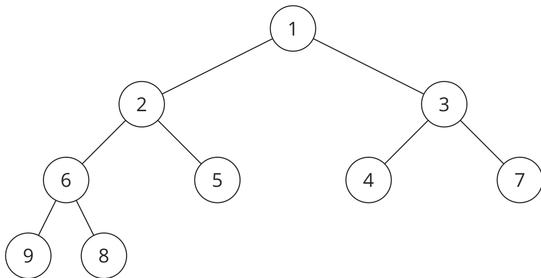
Dequeue

- “Remove” the root: Replace it by last element.
- Restore heap property: Call Min-Heapify(A, 0)



Dequeue

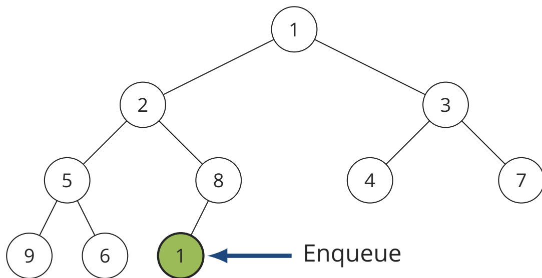
- “Remove” the root: Replace it by last element.
- Restore heap property: Call Min-Heapify(A, 0)
- $\mathcal{O}(\log n)$ time



Using a Heap – Enqueue

Enqueue

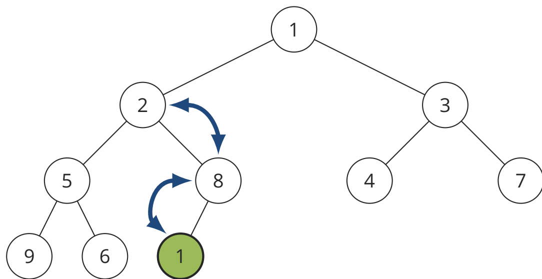
- Add new element at the end.



Using a Heap – Enqueue

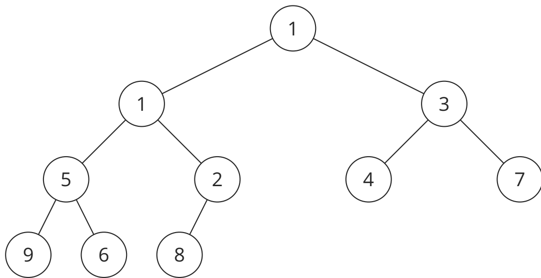
Enqueue

- Add new element at the end.
- Restore heap property: Exchange with parent until parent is smaller or equal.



Enqueue

- Add new element at the end.
- Restore heap property: Exchange with parent until parent is smaller or equal.
- $\mathcal{O}(\log n)$ time

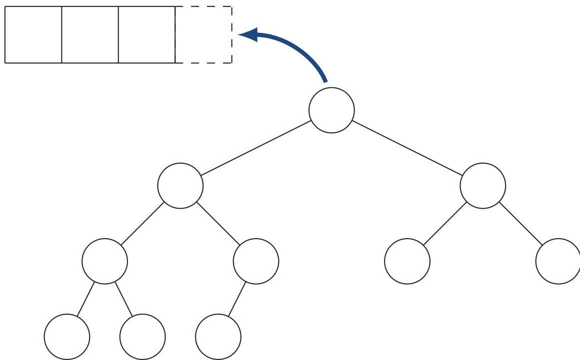


Heapsort

Heapsort

Idea

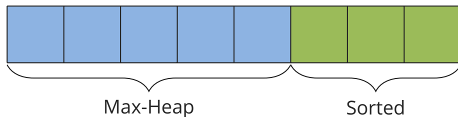
- Make array A to a heap and use it as priority queue.
- Order of removing is order in sorted array.



Heapsort

Algorithm

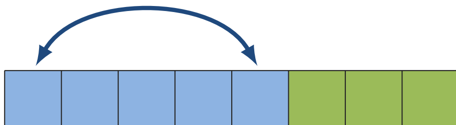
- Make array A to a max heap.



Heapsort

Algorithm

- Make array A to a max heap.
- Exchange $A[0]$ (root of the heap) with $A[\text{heapSize} - 1]$



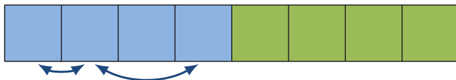
Heapsort

Algorithm

- Make array A to a max heap.
- Exchange $A[0]$ (root of the heap) with $A[\text{heapSize} - 1]$
- Decrease heapSize by 1

Algorithm

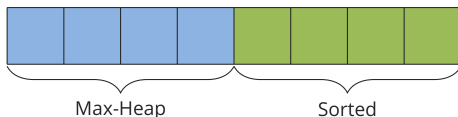
- Make array A to a max heap.
- Exchange $A[0]$ (root of the heap) with $A[\text{heapSize} - 1]$
- Decrease heapSize by 1
- Call Max-Heapify(A, 0).



Heapsort

Algorithm

- Make array A to a max heap.
- Exchange $A[0]$ (root of the heap) with $A[\text{heapSize} - 1]$
- Decrease heapSize by 1
- Call Max-Heapify(A, 0).



Heapsort

Properties

- Runtime: $\mathcal{O}(n \log n)$
- Memory: $\mathcal{O}(1)$ (if implemented correctly)
- Not stable

Exercises

Exercises

You wish to store a set of n numbers in either a max-heap or a sorted array. For each application below, state which data structure is better, or if it does not matter. Explain your answers.

(a) Want to find the maximum element quickly.

(b) Want to be able to delete an element quickly.

(c) Want to be able to form the structure quickly.

(d) Want to find the minimum element quickly.

Exercises

Give an $\mathcal{O}(n \log k)$ time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists.

Exercises

Design a data structure that supports the following two operations:

- **addNum(i)**
  Adds an integer to the data structure.
- **findMedian()**
  Returns the median of all elements so far.

Exercises

You are given a max-heap with n elements. Give an algorithm to find the k largest elements. You are allowed to destroy the heap. How fast is your algorithm?