Heaps Review

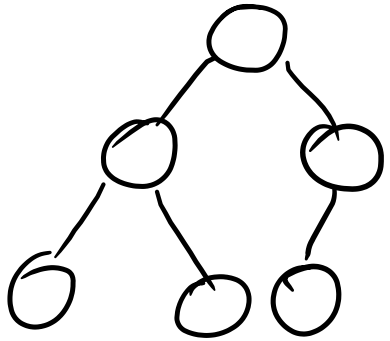
CSCI-104

A binary heap is ...

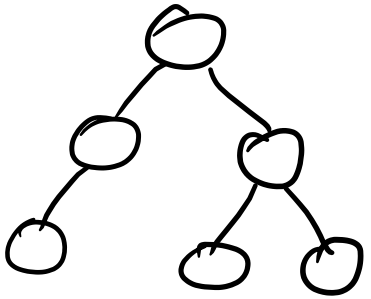
A binary heap is ...

a complete binary tree that satisfies the heap property

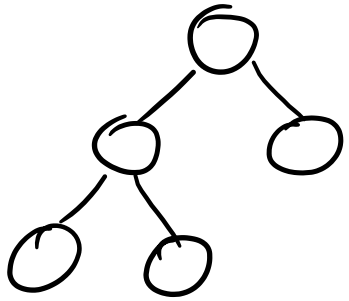
Completeness :



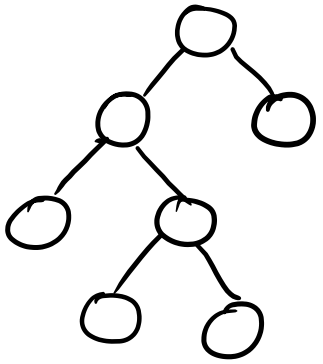
Complete



Incomplete



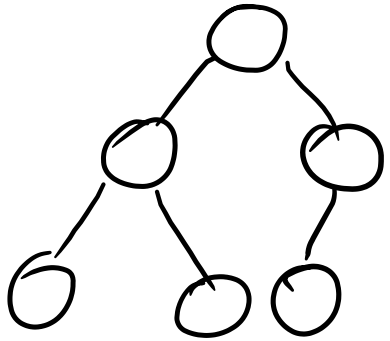
Complete



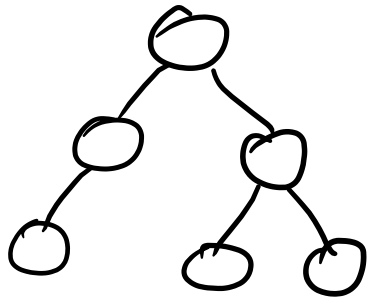
Incomplete

Completeness :

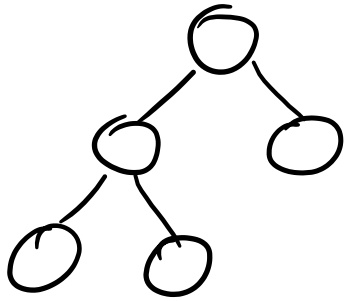
(Every level is full, except possibly the last level — if not full, must have all its node on the left)



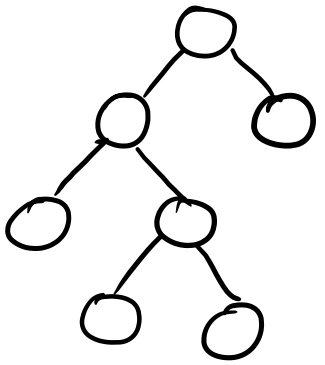
Complete



Incomplete



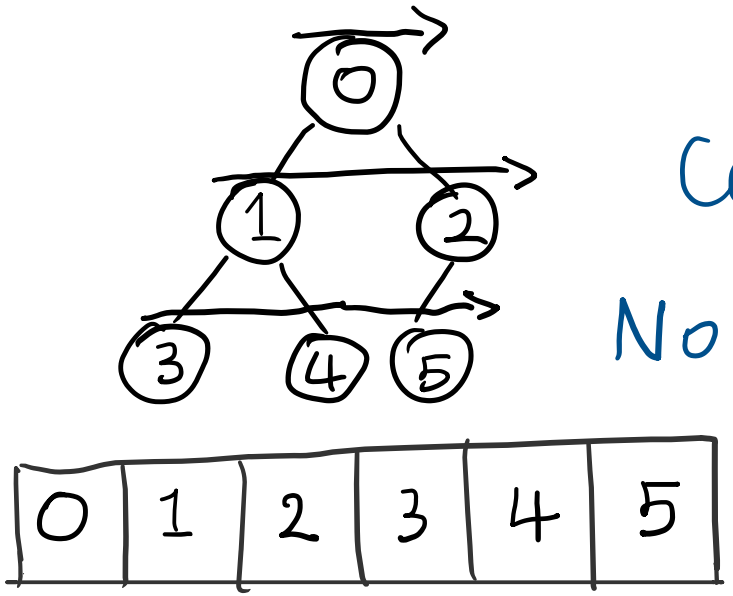
Complete



Incomplete

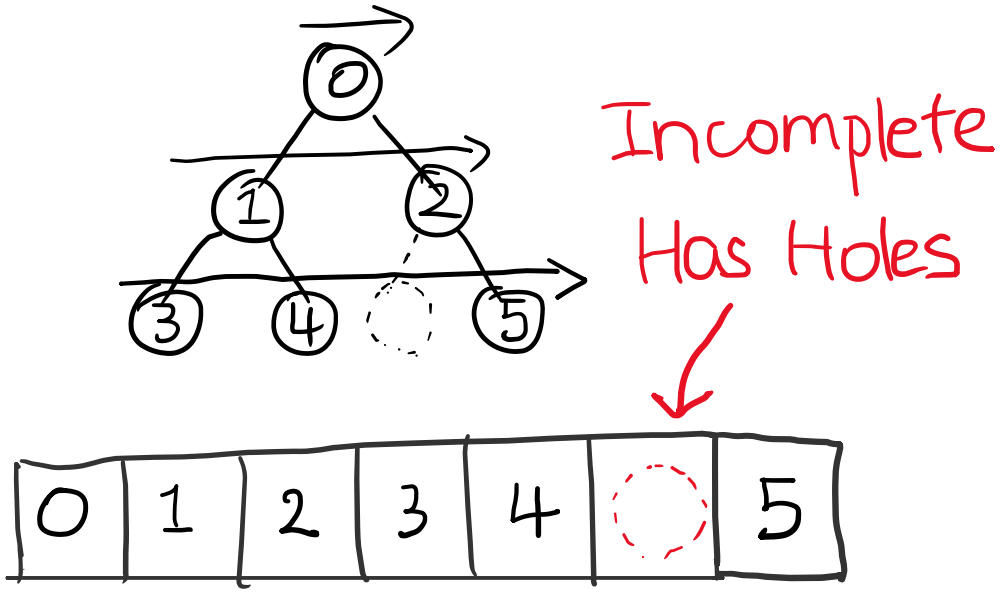
Completeness: (In practice)

You can store the tree in an array without any holes:



Complete

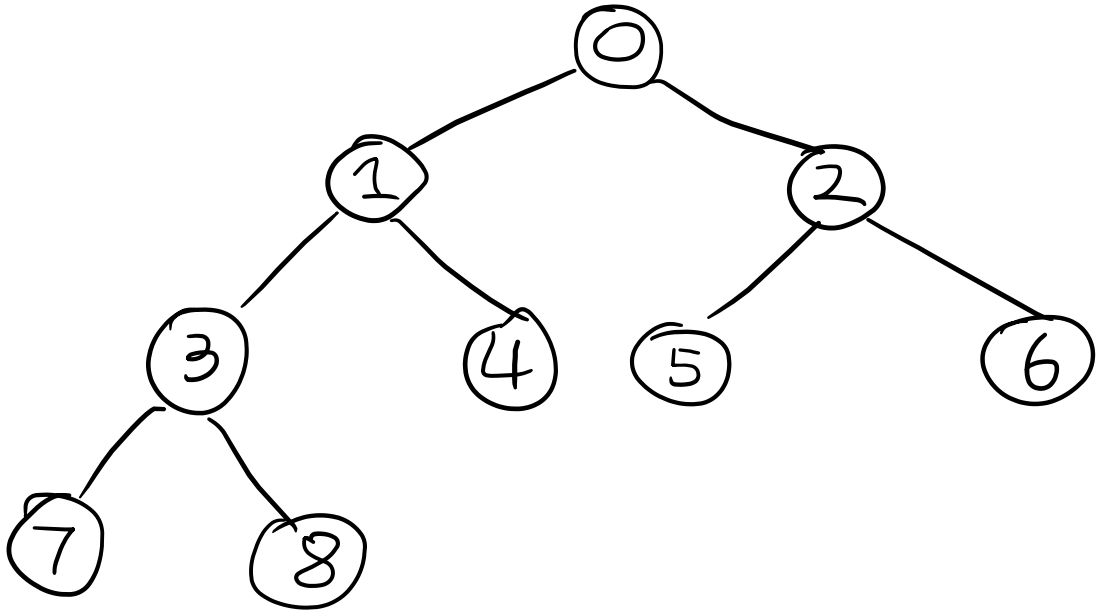
No Holes



Incomplete

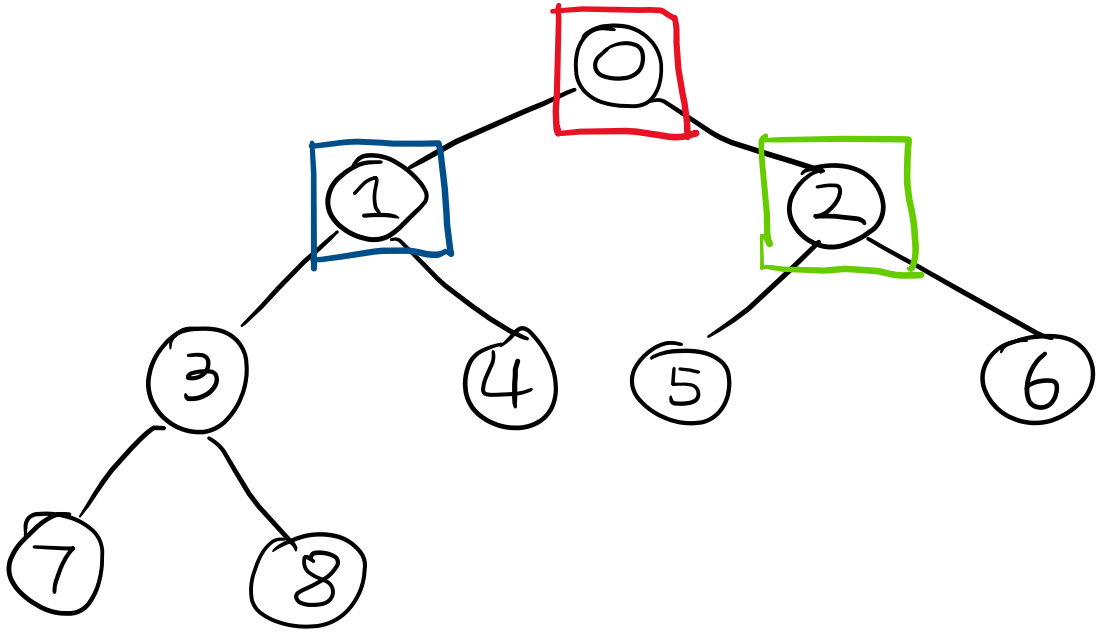
Has Holes

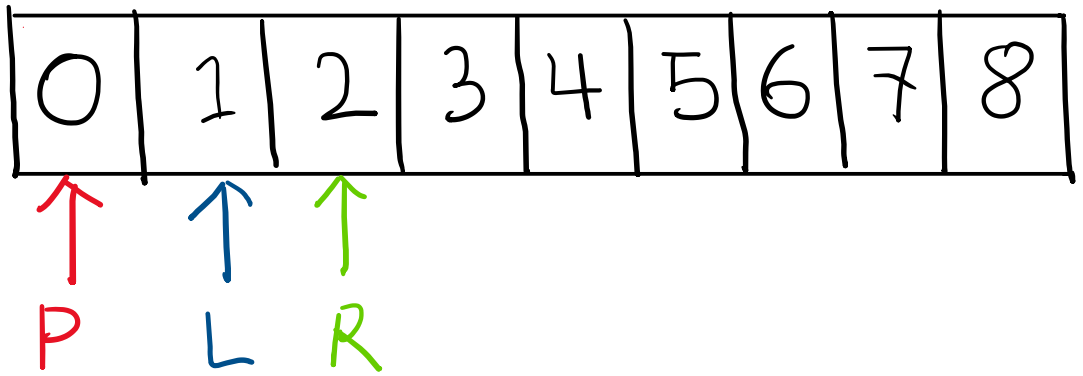
Finding Index of the children in an array



0	1	2	3	4	5	6	7	8

Finding Index of the children in an array



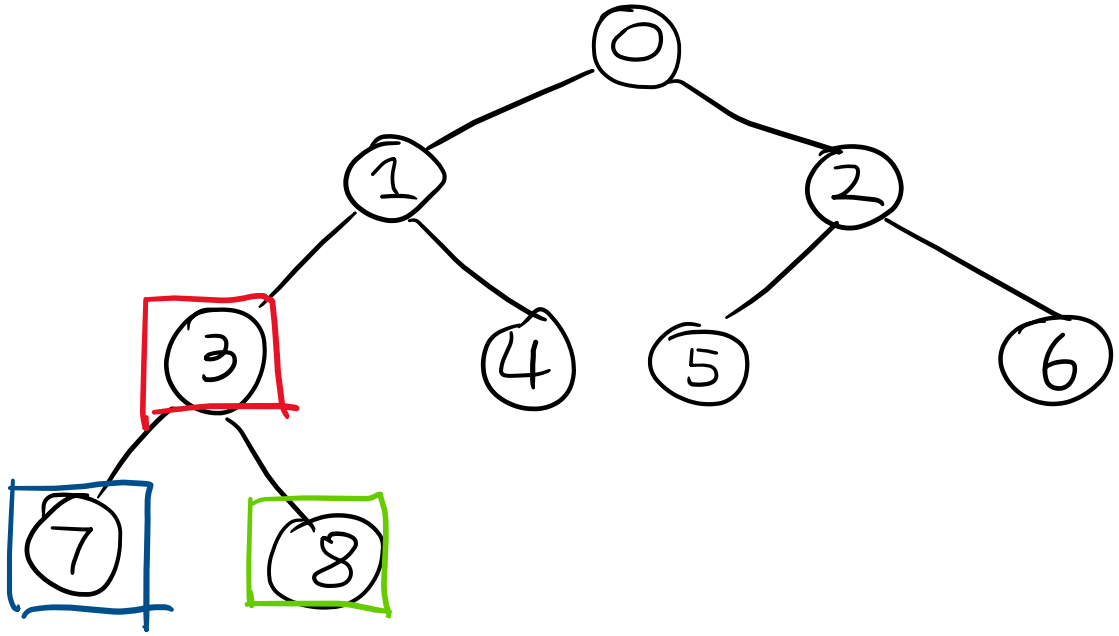


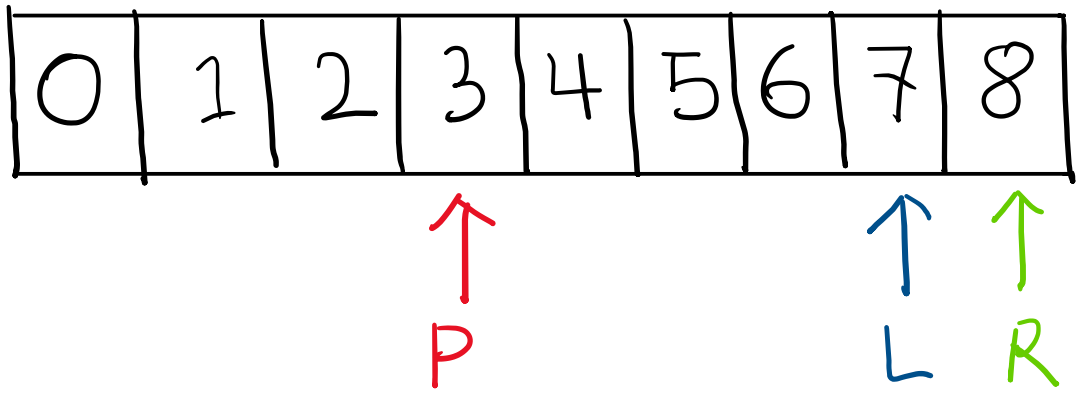
Parent = 0

Left = 1

Right = 2

Finding Index of the children in an array



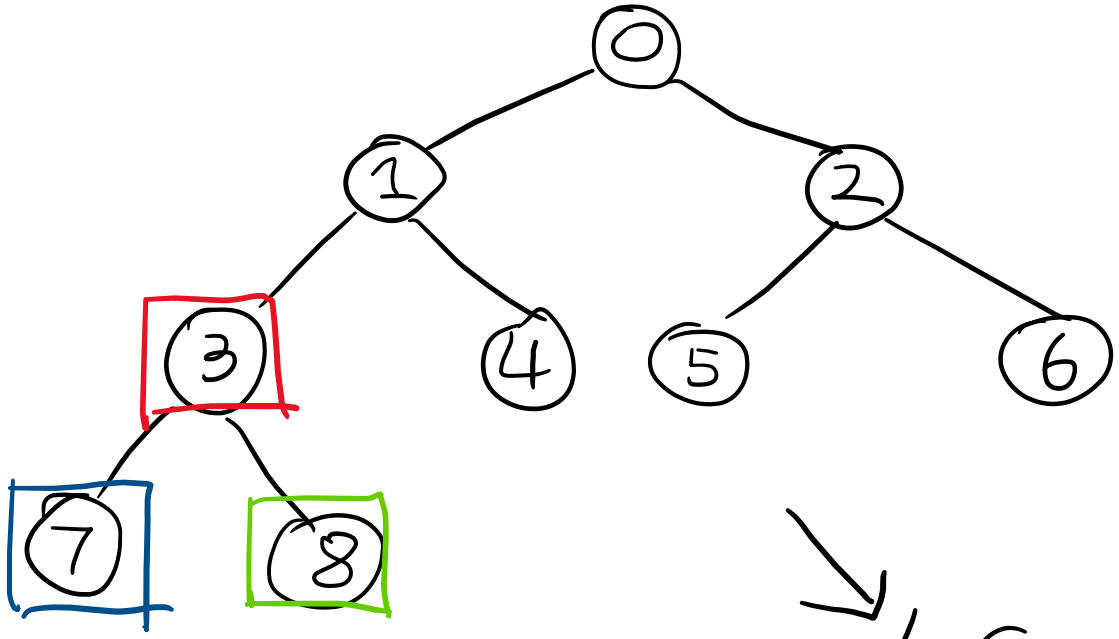


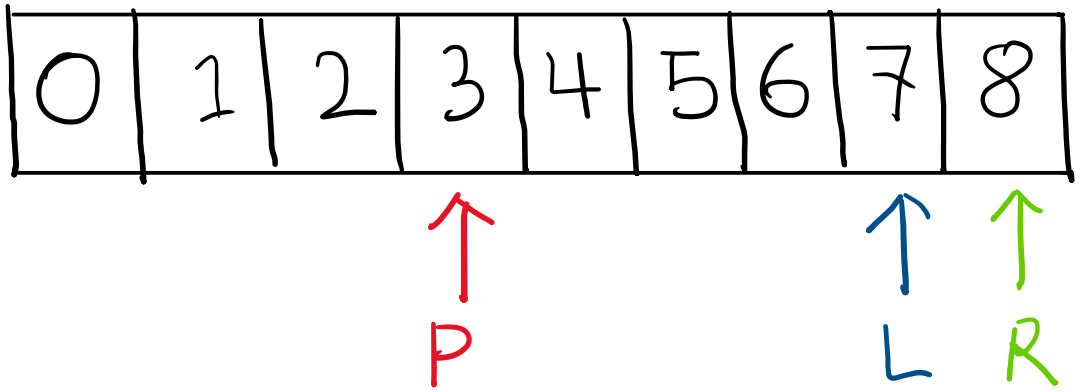
Parent = 3

Left = 7

Right = 8

Finding Index of the children in an array





Parent = 3

Left = 7

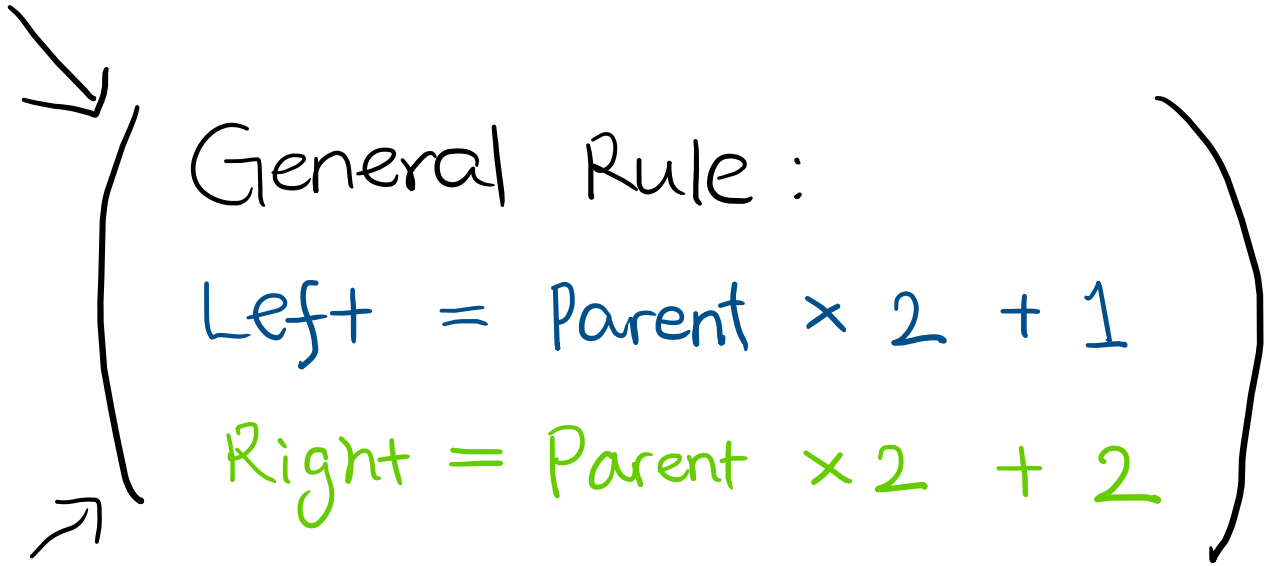
Right = 8

General Rule :

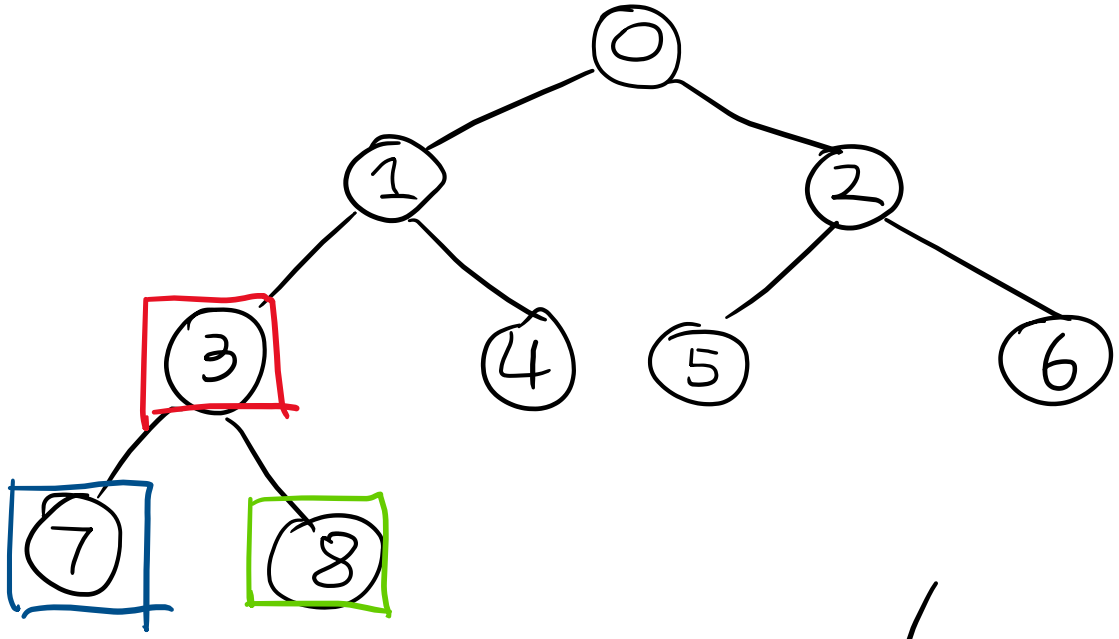
Left = Parent × 2 + 1

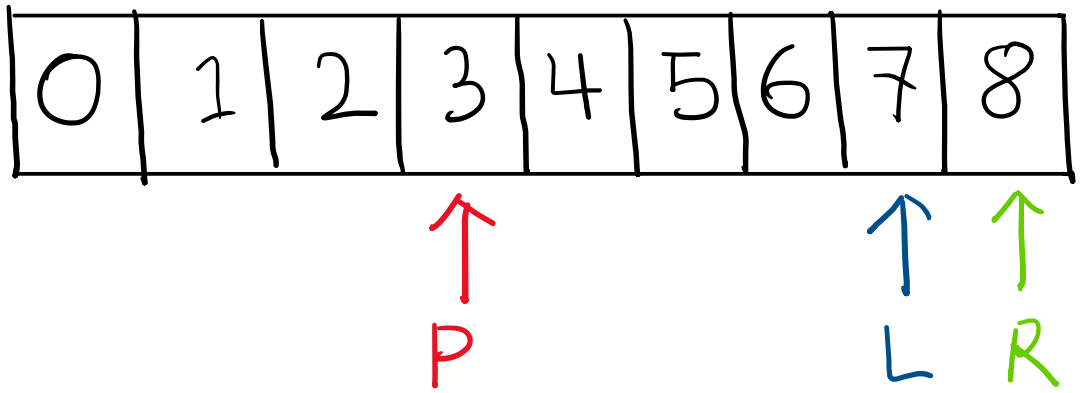
Right = Parent × 2 + 2

ONLY APPLIES IF ARRAY INDEX STARTS FROM 0.



Finding Index of the parent in an array





Parent = 3

Left = 7

Right = 8

(Parent = (child − 1) / 2

where "/" is C++ integer division)

The Heap Property :

Given a relational operator $\leftrightarrow$

(where $\leftrightarrow$ can be "$\leqslant$", "$\geqslant$", etc.)

The Heap Property :

Given a relational operator $\leftrightarrow$

(where $\leftrightarrow$ can be "$\leqslant$", "$\geqslant$", etc.)

For all element p in the heap,

$p \leftrightarrow x$, if x is in the left subtree of p

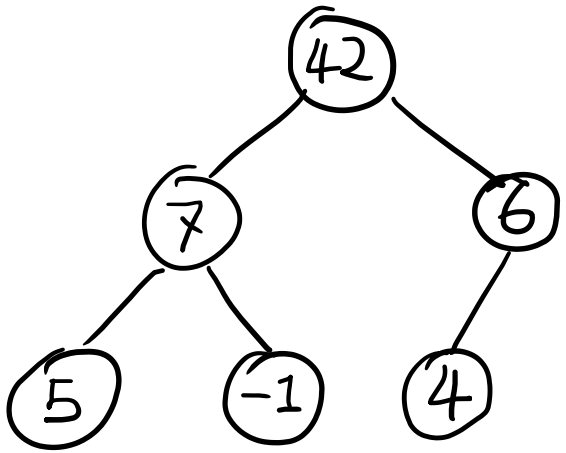
$p \leftrightarrow y$, if y is in the right subtree of p

In a max heap:

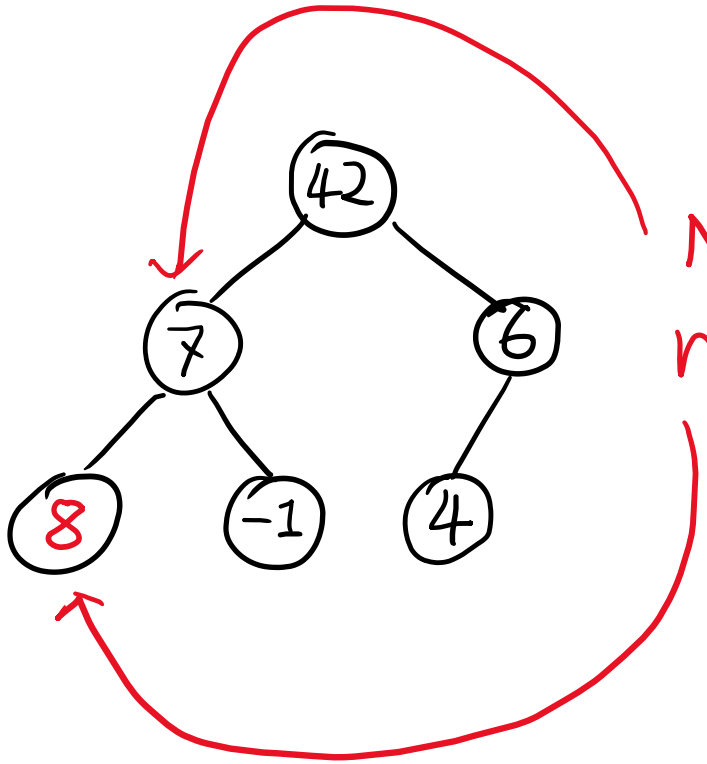
 is "$\geq$".

In a max heap:

(↔) is "$\geq$".



Is a max heap

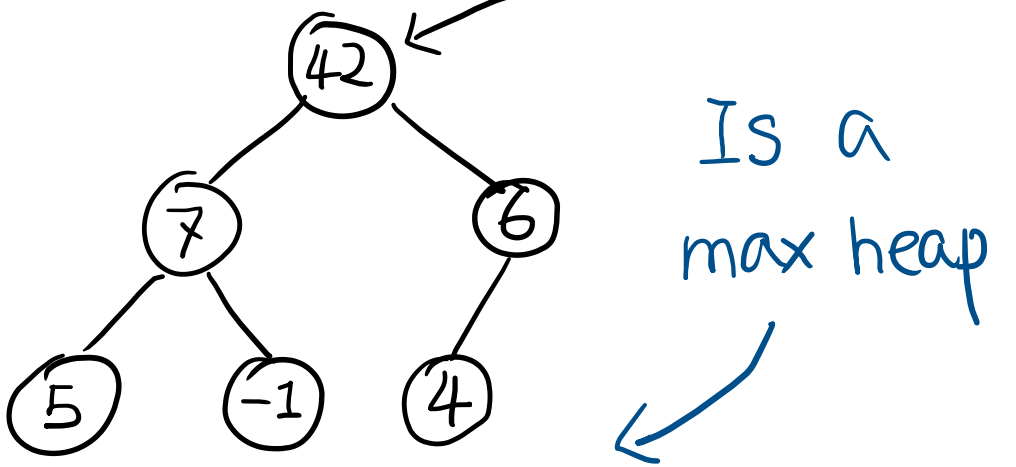


Not a max heap

(Since $7 \geq 8$ is false)

In a max heap:

(↔) is "≥".



Is a max heap

(Guarantees the root is the maximum)

Inserting into a heap

Step 1. Insert at the end of the array.

(i.e the rightmost position in the tree)

Inserting into a heap

Step 2. Swap the element with the parent

until "the parent ⟷ the element"

Removing from the heap

x : Element to remove

y : The element at the end of the array

Removing from the heap

x : Element to remove

y : The element at the end of the array

Step 1: Swap x and y

Removing from the heap

x : Element to remove

y : The element at the end of the array

Step 2: Pop x from the array.

Removing from the heap

x : Element to remove

y : The element at the end of the array

Step 3 : Swap y (now at the root) with its "larger" child until y is "larger" than its every child.

Removing from the heap

x : Element to remove

y : The element at the end of the array

(u is "larger" than v if $u \geq v$)

Step 3 : Swap y (now at the root) with its "larger" child until y is "larger" than its every child.

Can also be equal