Heaps Review

CSCI-104
A **binary** heap is ...
A binary heap is ...

a complete binary tree that satisfies the heap property
Completeness:

Complete Complete

Incomplete Incomplete
Completeness:

- Complete: Every level is full, except possibly the last level — if not full, must have all its nodes on the left.
- Incomplete:
Completeness: (In practice)

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

Complete
No Holes

Incomplete
Has Holes

```
0 1 2 3 4 5
```

```
0 1 2 3 4 ○ 5
```
Finding Index of the children in an array
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

\[
\begin{align*}
\text{General Rule:} & \\
\text{Left} & = \text{Parent} \times 2 + 1 \\
\text{Right} & = \text{Parent} \times 2 + 2
\end{align*}
\]
ONLY APPLIES IF ARRAY INDEX STARTS FROM 0.

General Rule:

\[
\begin{align*}
\text{Left} &= \text{Parent} \times 2 + 1 \\
\text{Right} &= \text{Parent} \times 2 + 2
\end{align*}
\]
Finding Index of the parent in an array

Parent = 3
Left = 7
Right = 8

\[
\text{Parent} = \left( \text{child} - 1 \right) / 2
\]

where "/" is C++ integer division
The Heap Property:

Given a relational operator $\leftrightarrow$

(where $\leftrightarrow$ can be "\( \leq \)", "\( \geq \)", etc.)
The Heap Property:

Given a relational operator \( \circ \)

(where \( \circ \) can be "\( \leq \)", "\( \geq \)", etc.)

For all element \( p \) in the heap,

\[ p \circ x, \text{ if } x \text{ is in the left subtree of } p \]

\[ p \circ y, \text{ if } y \text{ is in the right subtree of } p \]
In a max heap:

$\leftrightarrow$ is "\( \geq \)".
In a max heap:

- "\( \Rightarrow \) is " \( \geq \) ".

Is a max heap

Not a max heap

Since \( 7 \geq 8 \) is false
In a max heap:

is "\geq".

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

\((u \text{ is "larger" than } v \text{ if } u \leftrightarrow v)\)

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

\( \Rightarrow \) Can also be equal