# Heap

By Mani Abedii

A **heap** is a complete binary tree data structure that satisfies the **heap-order property**, which may be either:

- The **min-heap property**, where every parent node contains a key less than or equal to that of its children, or
- The **max-heap property**, where every parent node contains a key greater than or equal to that of its children.

Heaps are fundamental in the design of efficient priority queues and in the implementation of the heap sort algorithm.

## Structural Property

A heap is always a **complete binary tree**. Formally:

- All levels except possibly the last are fully populated.
- The last level is filled from left to right with no gaps.

The completeness property enables a compact array-based representation, eliminating the need for pointer-based tree nodes.

Given a node stored at index i (using 1-based indexing), its relatives are computed as:

- Parent: parent(i) = [i/2]
- Left child: left(i) = 2i
- Right child: right(i) = 2i + 1

This representation allows constant-time navigation between parents and children.

## Operations:

1. **Heapify-Up (Percolate Up):**

   Used during insertion.

   - Compares a node with its parent.
   - Swaps recursively until the heap-order property holds.

   This operation ensures newly inserted elements are placed correctly.

2. **Heapify-Down (Percolate Down):**
   Used during extraction or during build-heap.
   - Compares a node with its children.
   - Swaps with the child that maintains heap order.
   - Recurses until the node is in a valid position.

3. **Insertion:**

   Given a key x:

   - Insert x at the next available position (end of the array).
   - Perform the **heapify-up** operation.
     - Iteratively compare x with its parent and exchange them if the heap-order property is violated.
   - **Time Complexity: O(log(n))**
     - Because the height of a complete binary tree is Θ(log(n)).

4. **Extract-Min/Extract-Max:**

   To remove the root element:

   - Replace the root with the last element in the heap.
   - Reduce the heap size by one.
   - Apply the **heapify-down** operation:
     - Compare the root with its children and recursively exchange it with the child that preserves the heap-order property.
   - **Time Complexity: O(log(n))**

5. **Peek (Find-Min/Find-Max):**

   Returns the root element without removing it.

   - In a min-heap, this is the minimum key.
   - In a max-heap, this is the maximum key.
   - **Time Complexity: O(1)**

6. **Heapify (Build-Heap):**

   Given an arbitrary array of length n, a valid heap can be constructed by:

   - Identifying the lowest non-leaf node: [n/2]
   - Performing **heapify-down** from that node back to the root.

   This is known as the **Floyd Build-Heap** Algorithm.

   The Floyd-build algorithm runs in: **O(n)**

   Despite individual heapify-down costs up to O(log(n)), most operations occur near the leaves, reducing total cost.

## Priority Queues

Heaps implement priority queues. A priority queue is an abstract data type that stores a collection of elements, each associated with a priority, and supports efficient retrieval of the element with the highest or lowest priority.

Unlike regular queues (FIFO), the order of removal is determined by priority rather than insertion time.

Priority queues are fundamental in scheduling, graph algorithms, event simulation, and any system requiring dynamic ordering.

### Operations:

1. **Insertion:** Inserts an element into the priority queue.
2. **Find-Min/Find-Max:** Returns the highest-priority without removing it.
3. **Extract-Min/Extract-Max:** Removes and returns the element with the highest priority. This is the defining operation of a priority queue.
4. **Decrease-Key(i):** Lower the key (increase priority) of an element at index i.
5. **Delete:** Removes the element at position i.