Priority Queues and Heaps

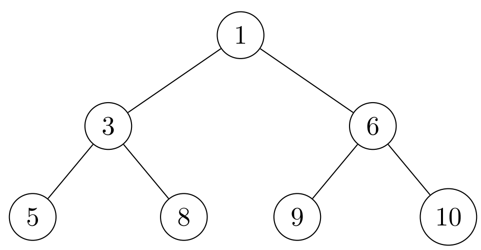
A **priority queue** is an abstract data structure where each element has a priority. Elements are served based on priority rather than just order of insertion.

Heaps are a concrete implementation of priority queues. A heap is a complete binary tree satisfying the *heap property*:

- **Min-Heap:** Parent $\leq$ children
- **Max-Heap:** Parent $\geq$ children

1 Use Case: Task Scheduling

A heap can represent task priorities, where lower numbers indicate higher priority (Min-Heap). The following figure shows a binary Min-Heap storing integers:



Min-Heap Property: Each parent node is less than or equal to its children.

- Root node = minimum element
- Efficient for implementing priority queues
- Balanced due to complete binary tree structure

2 Terminology

- **Heap:** Complete binary tree with ordering property
- **Priority:** Determines service order
- **Insert (Push):** Add element and restore heap
- **Extract (Pop):** Remove min/max and restore heap

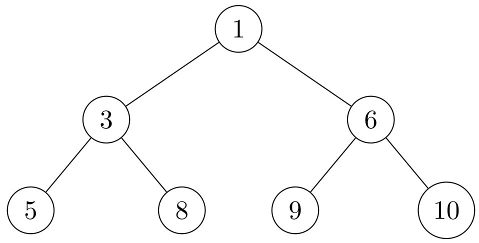
2.1 Array-Based Heap Representation

A binary heap can be stored efficiently using a **zero-based array** without using explicit pointers. Given a node at index i, its position relative to other nodes is determined by:

- **Left Child:** at index $2i + 1$

- **Right Child:** at index $2i + 2$
- **Parent:** at index $\left\lfloor \frac{i-1}{2} \right\rfloor$

Example: For the array `[1, 3, 6, 5, 8, 9, 10]` This corresponds to the following binary Min-Heap:



Benefits of Array Representation:

- Compact memory layout—no pointer overhead
- Easy index arithmetic for navigation
- Suited for cache-efficient access

Note: Array-based heaps are limited to complete binary trees. They must be restructured or resized carefully during insertions and deletions to maintain the heap shape and property.

3 C++ Implementation of a Min-Heap

```
#include <iostream>
#include <vector>
#include <stdexcept>
using namespace std;

class MinHeap {
  vector<int> heap;

  // Moves a node up the tree until the heap property is restored
  void heapifyUp(int i) {
    // While node is not the root and its value is less than its parent
    while (i > 0 && heap[i] < heap[(i - 1) / 2]) {
      // Swap the node with its parent
      swap(heap[i], heap[(i - 1) / 2]);
      // Move to the parent index
      i = (i - 1) / 2;
    }
  }

  // Moves a node down the tree until the heap property is restored
  void heapifyDown(int i) {
    int n = heap.size();
    while (2*i + 1 < n) { // While at least one child exists
      int left = 2*i + 1;
      int right = 2*i + 2;
      int smallest = left;

      // Choose the smaller of the two children
      if (right < n && heap[right] < heap[left])
        smallest = right;

      // If the current node is smaller than both children, stop
      if (heap[i] <= heap[smallest])
        break;
```

```
      // Otherwise, swap with the smaller child
      swap(heap[i], heap[smallest]);
      i = smallest; // Continue at the child index
    }
  }

public:
  void insert(int val) {
    heap.push_back(val);          // Insert at the end
    heapifyUp(heap.size() - 1);   // Restore heap property
  }

  int extractMin() {
    if (heap.empty()) throw runtime_error("Empty heap");
    int minVal = heap[0];         // Root has the minimum value
    heap[0] = heap.back();        // Replace root with last element
    heap.pop_back();              // Remove last element
    heapifyDown(0);               // Restore heap property
    return minVal;
  }

  int peek() const {
    if (heap.empty()) throw runtime_error("Empty heap");
    return heap[0];               // Return root without removing
  }

  bool empty() const { return heap.empty(); }
};
```

Usage:

```
MinHeap h;
h.insert(5);
h.insert(2);
h.insert(8);
cout << h.extractMin(); // prints 2
```

3.1 STL priority_queue

C++ STL provides a container adapter for priority queues backed by a heap.

```
#include <queue>
#include <vector>
using namespace std;

priority_queue<int> maxpq; // default: max-heap
priority_queue<int, vector<int>, greater<int>> minpq; // min-heap

minpq.push(4);
minpq.push(1);
cout << minpq.top(); // prints 1
minpq.pop();         // removes 1
```

4 Time Complexities

- **Insert:** $\mathcal{O}(\log n)$
- **Extract-Min/Max:** $\mathcal{O}(\log n)$
- **Peek:** $\mathcal{O}(1)$
- **Build-Heap (from array):** $\mathcal{O}(n)$

Why Are Heap Operations $O(\log n)$?

A binary heap is a complete binary tree stored as an array. Its height is $\log n$ (base 2), since each level doubles the number of nodes.

Operations like `insert`, `extract-min`, or `decrease-key` involve traversing from a leaf to the root or vice versa, adjusting positions to maintain the heap property. These traversals take at most $\log n$ steps, resulting in $O(\log n)$ time complexity.

5 Practice problem

Leetcode Problem 295: *Find median from data stream*
`https://leetcode.com/problems/find-median-from-data-stream/description/`