**Binary Heap**

* A **binary heap** (a kind of binary tree) is a data structure that is commonly used for priority queue implementations.
  + A heap is a concrete implementation of the priority queue abstract data type
* Binary heaps are usually just referred to as ***heaps****.* 
  + Unrelated to the memory pool used for dynamic allocation
* There are different kinds of heaps:
  + binary,
  + binomial,
  + Fibonacci
* We'll only talking about binary heaps
* Heaps make a tradeoff for node priority values instead of being good at other types of operations like searching, sorting in conventional binary search trees.

**Binary Heap Properties**

* A binary heap is a binary tree with two important properties that make it a good choice for priority queues:

1. **Completeness**
2. **Heap Order**

* An operation on a heap can disturb one of these properties, so a heap operation must not terminate until all heap properties are in order.

**1. Completeness**

* A *binary* *heap* is a *complete binary tree*.
* A complete binary tree has every level, except possibly the bottom, completely filled with nodes.
* If the bottom is not filled, the nodes in the last level are as **far left** as possible.

Chart, line chart

Description automatically generated

**2. Heap Order**

* There are generally two types of heaps:

1. **Minheap** – the root contains the item with the **smallest** value in the tree

A **minheap** is a ***complete binary tree*** whose **root** is either **empty** or

* 1. Contains a value **less than** **or equal to** the value in each of its children

and

* 1. Has heaps as its subtrees

In a minheap, for every node *X*, the key in the parent of *X* is smaller than (or equal to) the key in the current node *X*, with the exception of the root (which has no parent).

By the minheap-order property, the minimum element can always be found at the root.

Thus, we get the extra operation, findMin, in constant time.

1. **Maxheap** – places the item with the **greatest** value in its root

A **maxheap** is a ***complete binary tree*** whose **root** is either **empty** or

* 1. Contains a value **greater than** **or equal** to the value in each of its children

and

* 1. Has heaps as its subtrees

In a maxheap, for every node *X*, the key in the parent of *X* is greater than (or equal to) the key in the current node *X*, with the exception of the root (which has no parent).

By the maxheap-order property, the minimum element can always be found at the root.

Thus, we get the extra operation, findMax, in constant time.

* We will use minheaps for the rest of our examples.

**Heap Example 1**

* Here are simple examples of a maxheap and a minheap.

A picture containing diagram

Description automatically generated

* Remember that we can always find the max/min value in the tree at the root.

**Heap Example 2**

* The left tree is a valid minheap because it does not violate completeness and minheap order.
* In the following figure, the tree on the right is not a valid minheap because it violates the minheap order (the dashed line shows the violation of heap order).

A picture containing text, watch, map

Description automatically generated

**Binary Heap vs. Binary Search Tree**

* A binary heap is similar to a binary search tree, although it differs from a binary search tree in two significant ways:
  1. **Completeness**

While binary search trees come in many different shapes and sizes, heaps are always complete binary trees.

* 1. **Sorting (Heap Order)**

**T is a Binary Search Tree if:**

1. T is empty

or

1. Everything in its left subtree is smaller than it and everything in its right subtree is larger than it.

this is true if the left subtree and right subtree are binary trees.

Point 2 is what causes the really bad behavior in the worst case.

We probably don’t want exactly that requirement for implementing a priority queue.

While you can view a binary search tree as sorted, a heap is ordered in a much weaker sense. You cannot traverse a binary heap to output the items in sorted order.

**T is a Binary Heap if:**

1. T is a Complete Binary Tree
2. Every node is less than or equal to all of its children

this is true if the smallest element is in the root and results in no degenerate trees

* 1. **insert and removeMin**

**Binary Search Tree Worst Case:**

**insert = O(N)**

**removeMin = O(N)**

BSTs have really bad behavior in the worst case, but is it actually a common problem?

Fact: On average, the height of a BST is 𝑂(log 𝑛) (for some suitable definition of “average”)

Can we somehow enforce an O(log n) behavior in the worst case for priority queues?

Yes! This is what completeness + heap order accomplishes.

**Binary Heap Worst Case:**

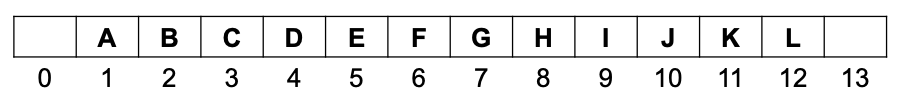
**insert = O(log N)**

**removeMin = O(log N)**

A binary heap will maintain a height of logn, which therefore results in a worst case runtime of O(log N) for these operations.

**An Array-Based Implementation of a Heap**

* A complete binary tree of height *h* has between 2*h* and 2*h+1* − 1 nodes.
* This implies that the height of a complete binary tree is ⌊log *N*⌋, which is *O*(log *N*).
* Because a heap is a complete binary tree, it can be represented in an array with no links necessary.
* This simpler array-based implementation saves memory because the elements in a complete binary tree are always contiguous.
* We can skip index 0 in the array to make the math simpler.
* Index 0 can be a good place to store the current size of the heap.
* Now, from node *i*, we can find
  + The left child at position 2i
  + The right child at position 2i + 1
  + The parent at position i / 2
* Thus, not only are links not required, but the operations required to traverse the tree are extremely simple

****

**Diagram

Description automatically generated with low confidence**