**Heaps Data Structure**

**Heaps**

* **Def**: It is a sub-division of Trees with additional properties known as the Heap properties

The Heap property:

The parent node will have a specific relationship with its children’s nodes. The relationship is either:

1. the parent node is greater than its children. This relationship is known as “Max Heap”

2. the parent node is smaller than its children. This relationship is known as “Min Heap”

Only one of the mentioned relationships apply for the entire Heap(every level in its structure or sub-tree).

Like Binary Search Trees BST, Heaps do require

Each node can only have at **most** two children

Unlike Binary Search Trees BST, Heaps **do** **not** require that:

1. All the children to the right must be greater than the parent node. And all children on the left must be smaller than the parent node. This order does not matter in Heaps, because Heaps are either Max Heap or Min Heap

2. All the children to the right must be greater than the children on the left. This order d**oes not** matter in Heaps as well

The insertion of nodes occurs from left to right. Therefore, we insert the left location first then if satisfied we insert the right location for each parental node. The most last element’s location to be filled in Heap bottom right. Below is an example of a faculty Heap. Because the rightest element location is filled before the left element in that node family

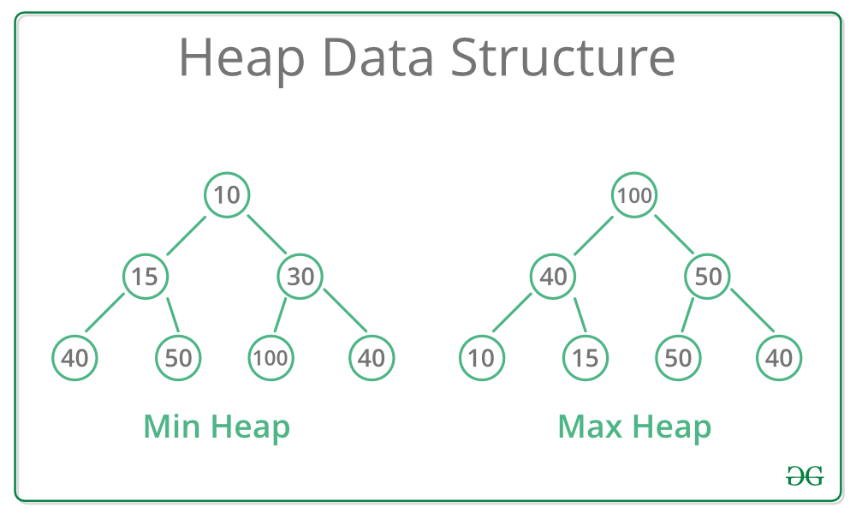
Diagram

Description automatically generated

In real life applications. Heaps are used in “propriety Queues”

Which are queues that have the ability to introduce tasks that are deemed priority and drops everything on focus on these tasks. A regular queue treats every task the same

The ability to skip in line within a queue based on the priority of the task



**Insert an array into a Heap**

A method to insert an array into a Heap is known as “Binary Heap”. This method follows few properties:

1. It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

This means. The first level must have 1 node. The second level must have 2 nodes. The third level must have 4 nodes and so on. Ofc this is if we have enough nodes. But we must strive to fill the Heap

2. A Binary Heap is either Min Heap or Max Heap. The same property must be recursively true (This means we won’t stop after we do one swap. We stop when the Heap is satisfied at every new element we insert from the array)

Exp1: The figure below presents the results of inserting an array into a heap. The array **none-sorted** array is [90,14,41,23,32,5,87,50,64,53]

The steps of insertion are:

1. Specify if you want the array to be a Max or Min Heap. The exp below is a Min Heap

2. Insert the first #, which is 5 at the top of the Heap

3. Insert the second # in the array. However, you must check if it satisfies the conditions of the Heap, you chosen

4. Always insert in the left element location then the right element location

If the Heap violates the condition, you must perform a “swap”

A swap is when you swap or exchange the child that violates the Heap property with its parent. Then swap again until the Heap is satisfactory

Diagram

Description automatically generated with medium confidence

**Insert an array into a Heap Run Time**

Heap’s insertion run time is O log(n)

This is because there is a set structure in the Heap, and it limits the number of operations the larger the Heap gets

The worst-case scenario is inserting a large element at the end of the Heap. Therefore, you would need to swap it with every parent element. This means each time you swap you move a while Heap level. Making your run time also O log(n)

**Check Max or Min Heap Run Time**

* This will be O(1) which is constant time. Because the Max or Min of a Heap is all the way on the top of the Heap

**Remove Max or Min Heap Run Time**

* The removal of the Max or Min. Will require us to swap the new empty space with the last inserted node. Then
* 1. swap the **last element inserted** with all the children nodes until the Heap’s structure is satisfied.
  + The last element inserted should be located on the rightest location from the bottom. This is because during each level insertion we insert from left to right. Without the consideration that either left or right elements are larger than each other
* The deletion or removal run time at worst case scenario is O(log(n))