Binary Heap Trees

Binary Heap Trees

Definition
A *binary heap tree* is a complete binary tree which is either empty or satisfies the following conditions:

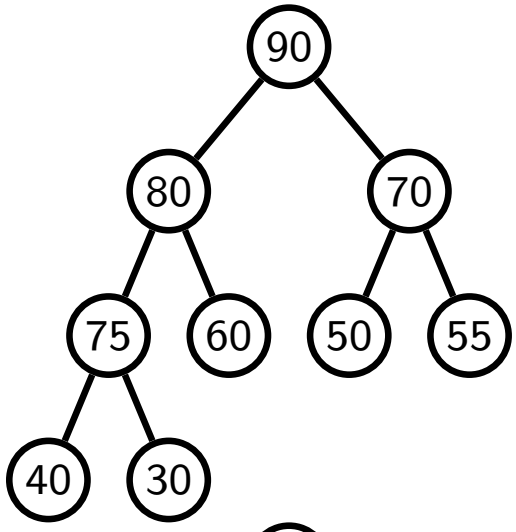
1. The priority of the root is greater than or equal to that of its children.
2. The left and right subtrees of the root are heap trees.

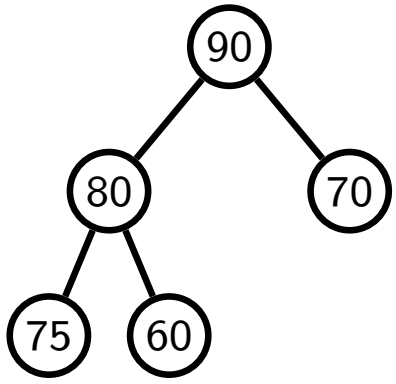
Note that, unlike in Binary Search Trees, there is no restriction on the relationship between the left and right children of any node: Binary Heap Trees do not keep the value stored in left child node less than that stored in the right child node.

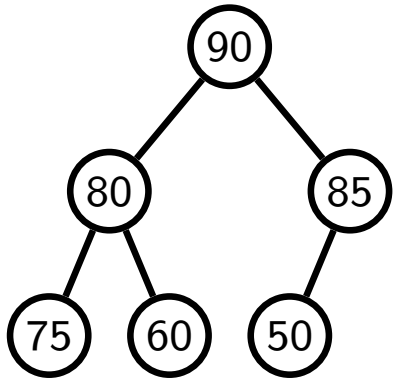
Thus we have no need to keep the Binary Heap Tree sorted, which makes working with the tree somewhat easier.

Binary Heap Tree Examples

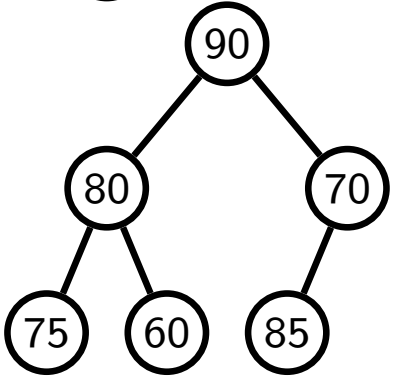
Valid:

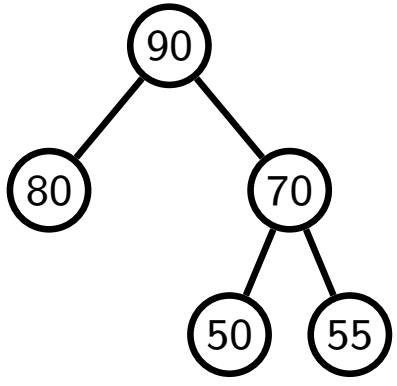


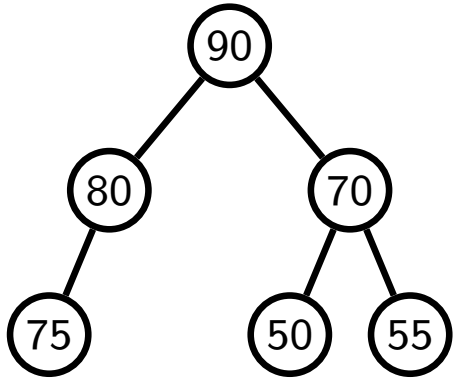




Invalid:







Priority Heap: Java-like Pseudo-Code

WARNING: Necessary error checking has been omitted!

```
1  public class PriorityHeap{
2    private int MAX = 100;
3    private int heap[MAX+1];
4    private int n = 0;
5
6    public int value(int i){
7      if (i < 1 or i > n)
8        throw IndexOutOfBoundsException;
9      return heap[i];
10   }
11   public boolean isRoot(int i) { return i == 1; }
12   public int level(int i)      { return log(i); }
13   public int parent(int i)     { return i / 2; }
14   public int left(int i)       { return 2 * i; }
15   public int right(int i)      { return 2 * i + 1; }
16   // More methods to be added here
17 }
```

Priority Heap: Java-like Pseudo-Code

```
1   public boolean isEmpty() {
2     return n == 0;
3   }
4
5   public int root() {
6     if ( heapEmpty() )
7         throw HeapEmptyException;
8      else return heap[1]
9   }
10
11  public int lastLeaf()) {
12     if ( heapEmpty() )
13        throw HeapEmptyException;
14     else return heap[n]
15  }
```