Binary Heap Implementation

Here is the reference code for the Binary Heap Implementation. Make sure to refer to the video lecture for the full explanation!

Binary Heap Operations

The basic operations we will implement for our binary heap are as follows:

- BinaryHeap() creates a new, empty, binary heap.
- insert(k) adds a new item to the heap.
- findMin() returns the item with the minimum key value, leaving item in the heap.
- delMin() returns the item with the minimum key value, removing the item from the heap.
- isEmpty() returns true if the heap is empty, false otherwise.
- size() returns the number of items in the heap.
- buildHeap(list) builds a new heap from a list of keys.

In [1]:
```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0


    def percUp(self,i):

        while i // 2 > 0:

            if self.heapList[i] < self.heapList[i // 2]:


                tmp = self.heapList[i // 2]
                self.heapList[i // 2] = self.heapList[i]
                self.heapList[i] = tmp
            i = i // 2

    def insert(self,k):

        self.heapList.append(k)
        self.currentSize = self.currentSize + 1
        self.percUp(self.currentSize)

    def percDown(self,i):

        while (i * 2) <= self.currentSize:

            mc = self.minChild(i)
            if self.heapList[i] > self.heapList[mc]:

                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[mc]
                self.heapList[mc] = tmp
            i = mc

    def minChild(self,i):
```

```
        if i * 2 + 1 > self.currentSize:

            return i * 2
        else:

            if self.heapList[i*2] < self.heapList[i*2+1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self):
        retval = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize = self.currentSize - 1
        self.heapList.pop()
        self.percDown(1)
        return retval

    def buildHeap(self,alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist[:]
        while (i > 0):
            self.percDown(i)
            i = i - 1
```