Implement a Heap data structure in Haskell according to the instructions below. Provide comments (including file headers with your name, date, homework number, etc.) and test showing that your heap functions are working correctly. For information about the heap structure (and some hints), see the heap notes on the course webpage as well as the example implementation of a binary search tree. You should treat this as a regular assignment (although its optional) in that you turn in your hardcopy with a cover sheet, tests showing your code works, a description of your testing approach, and a discussion of challenges and issues. Also please submit your assignment to the online submission site. Note that for full credit, you must consider a range of test cases for each function.

**Instructions.** Define an algebraic min-heap data type called `Heap` that consists of value constructors `Node` and `Nil`. Your `Heap` type should be an instance of both `Eq` and `Show`. Implement the following functions for `Heap`:

1. `insert :: Ord a => a -> Heap a -> Heap a`, which takes a value and a heap, and returns a new heap containing the value.
2. `getMin :: Heap a -> a`, which returns the min value in the given heap. If the heap is empty, print the error "Empty Heap".
3. `deleteMin :: Ord a => Heap a -> Heap a`, which returns a copy of the given heap with the min value removed.
4. `buildHeap :: Ord a => [a] -> Heap a -> Heap a`, which takes a List and a heap, and adds each element in the list to the heap.
5. `heapSort :: Ord a => [a] -> [a]`, which sorts a given list (from lowest to highest). Your `heapSort` function must use your heap implementation (i.e., build the heap from the list, and then repeatedly obtaining the minimum element to construct the sorted list).