```java
import java.util.Comparator;
import java.util.LinkedList;

public class Node<E>
{
    public int order;
    public Node<E> parent;
    public Comparator<E> comp;
    public LinkedList<Node<E>> children;
    public LinkedList<E> data;

    public Node(int theOrder, Comparator<E> theComp)
    {
        order = theOrder;
        comp = theComp;
        parent = null;
        children = new LinkedList<Node<E>>();
        data = new LinkedList<E>();
    }

    public Node(int theOrder, Comparator<E> theComp, Node<E> left, E item, Node<E> right)
    {
        this(theOrder, theComp);
        data.add(item);
        children.add(left);
        left.parent = this;
        children.add(right);
        right.parent = this;
        }

    public Node(int theOrder, Comparator<E> theComp, Node<E> theParent, LinkedList<E> theData,
LinkedList<Node<E>> theChildren)
    {
        order = theOrder;
        comp  = theComp;
        data  = theData;
        parent = theParent;
        children = theChildren;

        for(Node<E> child: children) {
            child.parent = this;
        }
    }

    public boolean hasOverflow()
    {
        return data.size() > order;
    }

    public boolean isLeaf()
    {
        return children.isEmpty();
    }

    public Node<E> childToFollow(E item)
    {
        if(!this.isLeaf()) {
```

```
57              int i = 0;
58              while(i < data.size()) {
59                  if (comp.compare(data.get(i), item) > 0) {
60                      break;
61                  }
62                  ++i;
63              }
64              return children.get(i);
65          }
66          return null;
67      }
68
69      public void leafAdd(E item)
70      {
71          int i = 0;
72
73          if(data.size()==0) {
74              data.addFirst(item);
75              return;
76          }
77
78          while(i < data.size()) {
79              if(comp.compare(data.get(i), item) > 0) {
80                  break;
81              }
82              ++i;
83          }
84          data.add(i, item);
85      }
86
87      public void split()
88      {
89          int midInd = data.size()/2;
90          if(parent != null) {
91              int ind = parent.children.indexOf(this);
92              E mid = data.get(midInd);
93
94              parent.data.add(ind, mid);
95              Node<E> sibling = new Node<>(order, comp, parent, new LinkedList<E>(), new
    LinkedList<Node<E>>());
96              parent.children.add(ind + 1, sibling);
97
98              sibling.data.addAll(data.subList(midInd+1, data.size()));
99              data.subList(midInd, data.size()).clear();
100
101             if(!children.isEmpty()) {
102                 sibling.children.addAll(children.subList(midInd+1, children.size()));
103                 children.subList(midInd, children.size()).clear();
104             }
105         }
106         else {
107             E mid = data.get(data.size()/2);
108             Node<E> sibling = new Node<>(order, comp);
109             Node<E> par = new Node<>(order, comp, this, mid, sibling);
110
111             sibling.data.addAll(data.subList(midInd+1, data.size()));
112
```

```
113             data.subList(midInd, data.size()).clear();
114         }
115     }
116
117     public boolean contains(E item)
118     {
119         return data.contains(item);
120     }
121 }
122
```