Sorted Lists

Chapter 16

Pre-requisite: Chapters 8, 12, 14
Motivation for Sorted List

- Consider the statement
  - Top 10, Top percentile, 2nd Quartile,…
- Faster computation of statistics
  - Minimum, Maximum, Median
  - Quartile, Decile, Percentile
  - Totals by quartile, deciles, percentile
- Faster algorithms for
  - Search - Binary Search algorithm
  - Duplicate elimination
  - Minimum Orthogonal Bounding Rectangle
  - …
Objectives

• Use sorted list in a program
• Describe differences between ADT list and ADT sorted list
• Implement ADT sorted list by using chain of linked nodes
• Implement ADT sorted list by using operations of ADT list

Contents

• Specifications for the ADT Sorted List
  ▪ Using the ADT Sorted List
• A Linked Implementation
  ▪ The Method \texttt{add}
  ▪ The Efficiency of the Linked Implementation
• An Implementation That Uses the ADT List
  ▪ Efficiency Issues
Sorted Lists

- We extend the capability of a list
  - Previous example used list to organize names
    - in alphabetical order
- Consider need to keep list sorted in
  - numerical or alphabetic order after list established
  - We add or remove an element
  - The ADT handles keeping elements in order
- Possible operations
  - For simplicity, duplicate entries allowed
  - Must determine where in list element is added
  - Can ask if list contains specified entry
  - Must be able to remove an entry
Abstract Data Type: Sorted List

• **Data**
  - A collection of objects in sorted order and having same data type
  - Number of objects in collection

• **Operations**
  - `add(newEntry)`
  - `remove(anEntry)`
  - `getPosition(anEntry)`
  - `getEntry(givenPosition)`
  - `contains(anEntry)`
  - `remove(givenPosition)`
  - `clear()`
  - `getLength()`
  - `isEmpty()`
  - `toArray()`
Abstract Data Type: Sorted List

```java
public interface SortedListInterface < T extends Comparable < ? super T > >
{
    public void add (T newEntry);
    public boolean remove (T anEntry);
    public int getPosition (T anEntry);

    // Inherit following methods from ADT List (Segment 12.7 of Chapter 12)
    public T getEntry (int givenPosition);
    public boolean contains (T anEntry);
    public T remove (int givenPosition);
    public void clear ()
    public int getLength ()
    public boolean isEmpty ()
    public T [] toArray ()
}
```
A Linked Implementation

- Linked implementation of the ADT sorted list, Listing 16-2
- Note different versions of method add
  - Iterative version
  - Recursive version
Figure 16-1 Places to insert names into a sorted chain of linked nodes

firstNode

Ally < Bob  Cathy < Jill  Luke < Mike  Sue == Sue  Tom > Sue
Figure 16-2 Recursively adding *Luke* to a sorted chain of names


```
findFirstNode

Bob -> Jill -> Mike -> Sue
```


```
. . . -> Jill -> Mike -> Sue
```

*Luke < Mike*, so add *Luke* here, at the beginning of the rest of the chain

```
. . . -> Mike -> Sue
```
Linked Implementation of Sorted List

```java
public class SortedLinkedList<T> implements SortedListInterface<T> {
    private Node firstNode;
    private int numberOfEntries;
    public SortedLinkedList() { firstNode = null; numberOfEntries = 0; }
    public void add(T newEntry) {
        firstNode = add(newEntry, firstNode);
        numberOfEntries++;
    }
    // Recursive definition of add(,), pp. 420
    private Node add(T newEntry, Node currentNode) {
        if ((currentNode == null) || (newEntry.compareTo(currentNode.getData()) <= 0)) {
            currentNode = new Node(newEntry, currentNode);
        } else {
            Node nodeAfter = add(newEntry, currentNode.getNextNode());
            currentNode.setNextNode(nodeAfter);
        }
        return currentNode;
    }
    // implementations of other methods as needed …
}
```
Figure 16-3 Recursively adding a node at the beginning of a chain

(a) The list before any additions

(b) As add("Ally", firstNode) begins execution

(c) After a new node is created (the base case)

The private method returns the reference that is in currentNode

(d) After the public add assigns the returned reference to firstNode
Figure 16-4 Recursively adding a node between existing nodes in a chain

(a) As the recursive call `add("Luke", currentNode.nextNode())` begins execution

(b) After a new node is created (the base case)

(c) After the returned reference is assigned to `nodeAfter`

(d) After `currentNode.nextNode(nodeAfter)` executes
Exercises

Q. A. What is the worst-case time-complexity of add() for a list with N items?

Q. 5. Consider add() method. If the entry is already in the list, where will add method insert it in the list?
   (a.) before the first occurrence of the entry
   (b.) after the last occurrence of the entry
   (c.) somewhere else

Q. 6. What would be the answer to previous question if you changed <= to < in the conditional invoking compareTo() method?
## Efficiency of the Linked Implementation

![Table showing worst-case efficiencies of operations on ADT sorted list for two implementations](image)

<table>
<thead>
<tr>
<th>ADT Sorted List Operation</th>
<th>Array</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(newEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>getPosition(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>getEntry(givenPosition)</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>contains(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove(givenPosition)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>toArray()</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>clear(), getLength(), isEmpty()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

**FIGURE 16-5** The worst-case efficiencies of the operations on the ADT sorted list for two implementations
public class SortedLinkedList < T extends Comparable < ? super T > >
    implements SortedListInterface < T > {

    private Node firstNode;  private int numberOfEntries;

    public SortedLinkedList () { firstNode = null;  numberOfEntries = 0; }

    public void add (T newEntry) {
        Node newNode = new Node (newEntry);
        Node nodeBefore = getNodeBefore (newEntry);
        if (isEmpty () || (nodeBefore == null)) {
            newNode.setNextNode (firstNode);  firstNode = newNode;
        }  else  {  Node nodeAfter = nodeBefore.getNextNode ();
            newNode.setNextNode (nodeAfter); nodeBefore.setNextNode (newNode);
        }
        numberOfEntries++;
    }

    private Node getNodeBefore (T anEntry)  {
        Node currentNode = firstNode;    Node nodeBefore = null;
        while ((currentNode != null) && (anEntry.compareTo (currentNode.getData ()) > 0)) {
            nodeBefore = currentNode;  currentNode = currentNode.getNextNode ();
        }
        return nodeBefore;
    }
}
Implementation That Uses the ADT List

• Sorted list a natural application for ADT list
• Possible ways to reuse ADT list
  ▪ Use list as data field within class that implements sorted list
  ▪ Use inheritance to derive sorted list from list
• Our class SortedList will implement the interface SortedListInterface
• View source code, Listing 16-A
public class SortedList<T extends Comparable<? super T>> implements SortedListInterface<T> {

    private ListInterface<T> list;
    public SortedList() { list = new LList<T>(); }

    public void add(T newEntry) {
        int newPosition = Math.abs(getPosition(newEntry));
        list.add(newPosition, newEntry);
    }

    public boolean remove(T anEntry) {
        boolean result = false; int position = getPosition(anEntry);
        if (position > 0) { list.remove(position); result = true; }
        return result;
    }

    public int getPosition(T anEntry) {
        int position = 1; int length = list.getLength();
        while ((position <= length) && (anEntry.compareTo(list.getEntry(position)) > 0))
            { position++; }
        if ((position > length) || (anEntry.compareTo(list.getEntry(position)) != 0))
            { position = -position; }
        return position;
    }

    public T getEntry(int givenPosition) { return list.getEntry(givenPosition); }
}
Q. A. What is the worst-case time-complexity of add() for a list with N items?

Q. 9. Consider add() method. If the entry is already in the list, where will add method insert it in the list?
   (a.) before the first occurrence of the entry
   (b.) after the last occurrence of the entry
   (c.) somewhere else

Q. 10. Can a client of SortedList invoke add(position, entry) of ADT list? Why?

Q. 11. If a sorted list contains five duplicate objects. What is deleted by remove()?
   (a.) first occurrence of item, (b.) last occurrence, (c.) all occurrences

Q. 12 Assume that a sorted list named slist contains four names: Brenda, Carlos, Sarah, and Tom as strings. What will getPosition() return when anEntry represents:
   (a.) Carlos  (b.) Allen  (c.) Wendy  (d.) Tom  (e.) Jamie
Figure 16-6 An instance of a sorted list that contains a list of its entries

Figure 16-7 A sorted list in which Jamie belongs after Carlos but before Sarah
### ADT List Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getEntry(given Position)</code></td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>add(new Position, new Entry)</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>remove(given Position)</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>contains(anEntry)</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>clear(), getLength(), isEmpty()</code></td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

### ADT Sorted List Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(new Entry)</code></td>
<td>O(n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td><code>remove(anEntry)</code></td>
<td>O(n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td><code>getPosition(anEntry)</code></td>
<td>O(n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td><code>getEntry(given Position)</code></td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>contains(anEntry)</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>remove(given Position)</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><code>clear(), getLength(), isEmpty()</code></td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
End

Chapter 16