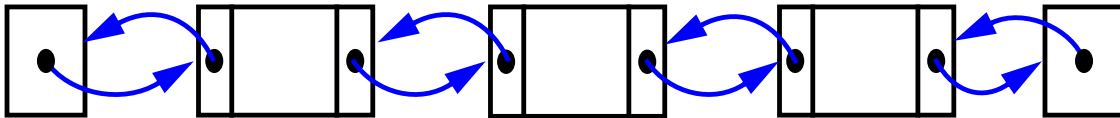


SEQUENCES

- Ranked Sequences
- Positions
- Positional Sequences
- General Sequences
- Bubble Sort Algorithm



The Ranked Sequence ADT

- A ranked sequence S (with n elements) supports the following methods:
 - **elemAtRank(r):**
Return the element of S with rank r ; an error occurs if $r < 0$ or $r > n - 1$
Input: Integer; *Output:* Object
 - **replaceElemAtRank(r,e):**
Replace the element at rank r with e and return the old element; an error condition occurs if $r < 0$ or $r > n - 1$
Input: Integer r , Object e ; *Output:* Object
 - **insertElemAtRank(r,e):**
Insert a new element into S which will have rank r ; an error occurs if $r < 0$ or $r > n - 1$
Input: Integer r , Object e ; *Output:* Object
 - **removeElemAtRank(r):**
Remove from S the element at rank r ; an error occurs if $r < 0$ or $r > n - 1$
Input: Integer; *Output:* Object

Array-Based Implementation

- Some Pseudo-Code:

Algorithm insertElemAtRank(r, e):

for $i = n - 1, n - 2, \dots, r$ **do**

$S[i+1] \leftarrow s[i]$

$S[r] \leftarrow e$

$n \leftarrow n + 1$

Algorithm removeElemAtRank(r):

$e \leftarrow S[r]$

for $i = r, r + 1, \dots, n - 2$ **do**

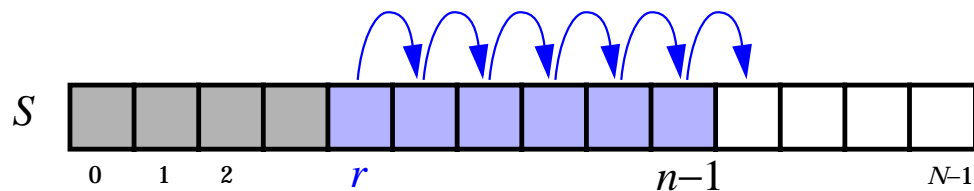
$S[i] \leftarrow S[i + 1]$

$n \leftarrow n - 1$

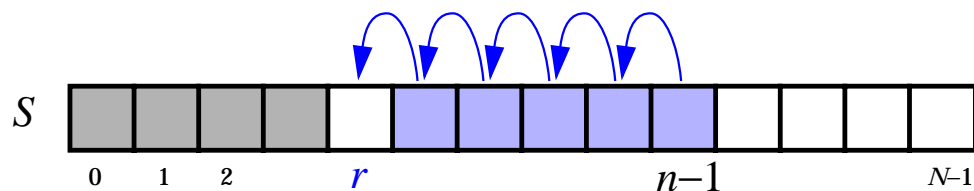
return e

- A Graphical Representation

insertElemAtRank(r, e):



removeElemAtRank(r):



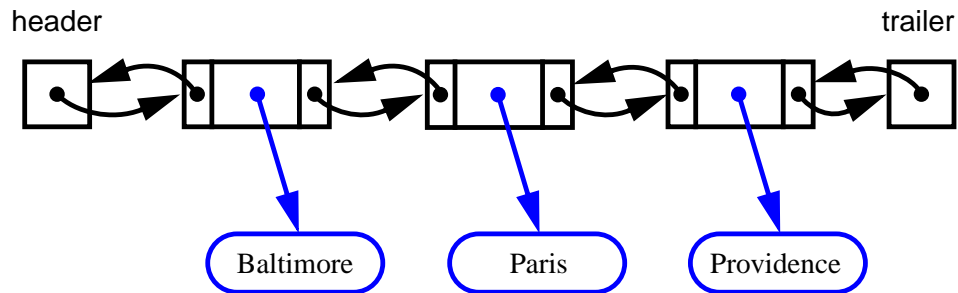
Array-Based Implementation (contd.)

- Time complexity of the various methods:

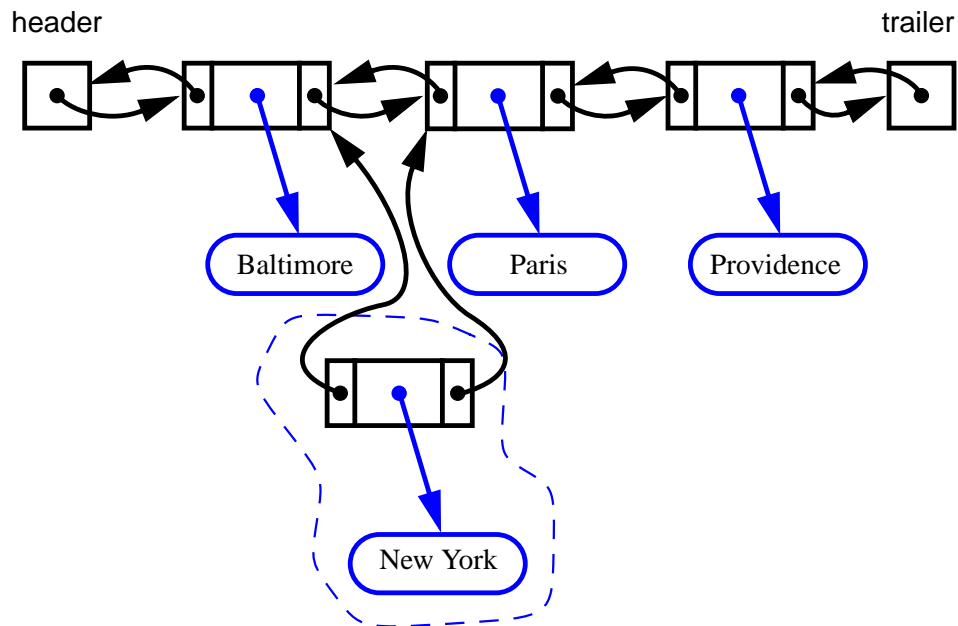
| Method | Time |
|-------------------|-------------|
| size | $O(1)$ |
| isEmpty | $O(1)$ |
| elemAtRank | $O(1)$ |
| replaceElemAtRank | $O(1)$ |
| insertElemAtRank | $O(n)$ |
| removeElemAtRank | $O(n)$ |

Implementation with a Doubly Linked List

- the list before insertion:

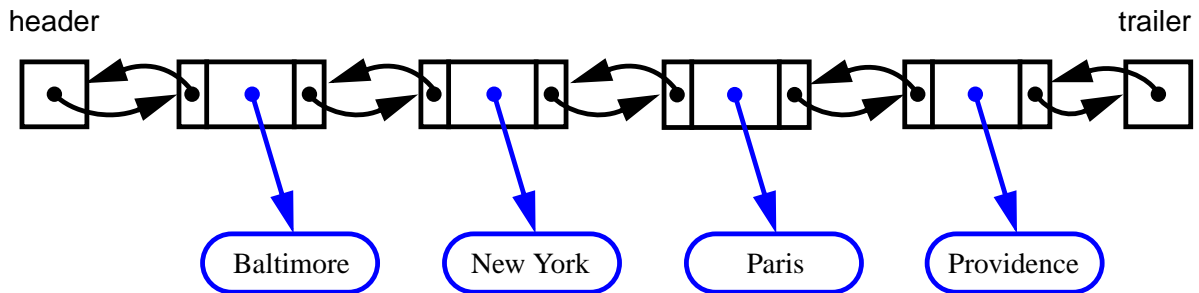


- creating a new node for insertion:

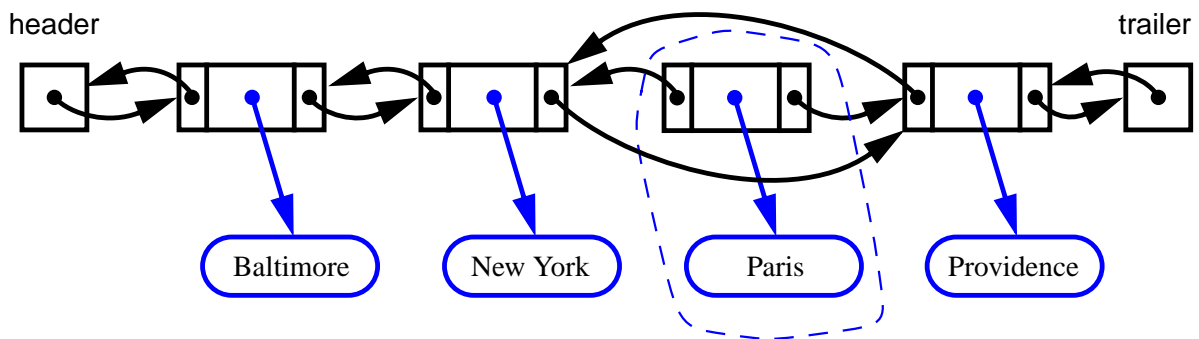


Implementation with a Doubly Linked List

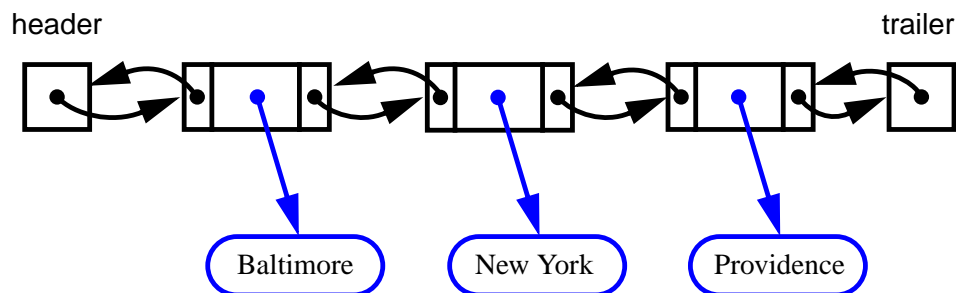
- the list after insertion and before deletion:



- deleting a node:



- after deletion:



Java Implementation

```
public class NodeRankedSequence
  extends MyDeque implements Deque,
  RankedSequence{
public void insertElemAtRank (int rank, Object element)
  throws BoundaryViolationException {
    if (rank != size()) //rank size() is OK for
                        //insertion
      checkRank(rank);
    DLNode next = nodeAtRank(rank); // the new node
                                    //will be right before this
    DLNode prev = next.getPrev(); // the new node
                                    //will be right after this
    DLNode node = new DLNode(element, prev, next);
    next.setPrev(node);
    prev.setNext(node);
    size++;
  }
public Object removeElemAtRank (int rank)
  throws BoundaryViolationException {
    checkRank(rank);
    DLNode node = nodeAtRank(rank); // node to
                                    //be removed
    DLNode next = node.getNext(); //node before it
```

Java Implementation (cont.)

```
DLNode prev = node.getPrev(); // node after it
prev.setNext(next);
next.setPrev(prev);
size--;
return node.getElement();
}

private DLNode nodeAtRank (int rank) {
    // auxiliary method to find the node of the
    //element with the given rank
    DLNode node;
    if (rank <= size()/2) { //scan forward from head
        node = header.getNext();
        for (int i=0; i < rank; i++)
            node = node.getNext();
    }
    else { // scan backward from the tail
        node = trailer.getPrev();
        for (int i=0; i < size()-rank-1 ; i++)
            node = node.getPrev();
    }
    return node;
}
```


Nodes and Positions

- Node-Based Operations:
 - Node specific methods, e.g. `removeAtNode(Node v)` and `insertAfterNode(Node v, Object e)`, would be $O(1)$.
 - However, node-based operations are not meaningful in an array-based implementation because there are no nodes in an array.
 - **Dilemma:**
 - If we do not include the-node based operations int the generic sequence ADT, we are not taking full advantage of doubly-linked lists.
 - If we do include them, we violate the generality of object oriented design.

Nodes and Positions (cont.)

- Positions:
 - Intuitive notion of “place” of an element.
 - This concept allows us to enjoy doubly-linked list without violating object-oriented design.
 - Positions have 2 methods:
 - element(): Return the element at the Position
Input: none; Output: Object
 - container(): Return the sequence that contains this position.
Input: none; Output: sequence
 - Positions are defined relatively.
 - Positions are not tied to an element or rank
 - A Sequence is a container of elements that are each stored in a position

The Positional Sequence ADT

- The methods are:
 - first()
 - last()
 - before()
 - after()
 - size()
 - isEmpty()
 - replace(p,e)
 - swap(p, q)
 - insertFirst(e)
 - insertLast(e)
 - insertBefore(p,e)
 - insertAfter(p,e)
 - remove(p)
 - isFirst(p)
 - isLast(p)

Doubly Linked List Implementation

- Implementation of a node using Positions

```
class NSNode implements Position {  
    private NSNode prev, next; // References to the  
                               //nodes before and after  
    private Object element; //Element stored in this  
                               //position  
    private Container cont; //Container of this  
                               //position  
    NSNode(NSNode newPrev, NSNode newNext,  
            Container container, Object elem) { //Initialize  
                                               //the node  
  
        prev = newPrev;  
        next = newNext;  
        cont = container;  
        element = elem;  
    }  
    public Container container()  
        throws InvalidPositionException {  
        if (cont == null)  
            throw new InvalidPositionException  
                ("Position has no container!");  
    }  
}
```

Doubly Linked List Implementation(cont.)

```
        return cont;
    }
    public Object element()
        throws InvalidPositionException {
        if (cont == null)
            throw new InvalidPositionException
                ("Position has no container!");
        return element;
    }
    // Acceser methods
    NSNode getNext() { return next; }
    NSNode getPrev() { return prev; }
    void setNext(NSNode newNext) { next = newNext; }
    // Update methods
    void setPrev(NSNode newPrev) { prev = newPrev; }
    void setElement(Object newElement) { element =
                                                newElement; }
    void setContainer(Container newCont) { cont =
                                                newCont; }
}
```

Doubly Linked List Implementation

- Code for other methods of a Doubly Linked List:

- checkPosition

```
protected NSNode checkPosition(Position p) throws  
    InvalidPositionException{  
    if (p==head)  
        throw new InvalidPositionException("Head of  
            the sequence is not a valid position");  
    if (p==tail)  
        throw new InvalidPositionException ("Tail of the  
            sequence is not a valid position");  
    }
```

- first

```
public Position first() throws  
    EmptyContainerException {  
    if(isEmpty())  
        throw new EmptyContainerException  
            ("Sequence is empty");  
    return head.getNext();  
    }
```

Doubly Linked List Implementation (cont.)

- before

```
public Position before(Position p) throws  
    InvalidPositionException, BoundaryViolationException{  
    NSNode n = checkPosition(p);  
    NSNode prev = n.getPrev();  
    if(prev==head)  
        throw new BoundaryViolationException (“Cannot  
            go past the beginning of the sequence”);  
    return prev;  
}
```

- insertAfter

```
public Position insertAfter (Position p, Object element)  
    throws InvalidPositionException{  
    NSNode n = checkPosition(p);  
    numElts++;  
    NSNode newNode = new NSNode(n, n.getNext(),  
        this, element);  
    n.getNext().setPrev(newNode);  
    n.setNext(newNode);  
    return newNode;  
}
```

Doubly Linked List Implementation (cont.)

- remove

```
public Object remove(Position p) throws  
    InvalidPositionException {  
        NSNode n = checkPosition(p);  
        numElts--;  
        NSNode nPrev = n.getPrev();  
        NSNode nNext = n.getNext();  
        nPrev.setNext(nNext);  
        nNext.setPrev(nPrev);  
        Object nElem = n.element();  
        // unlink the position from the list  
        //and make it invalid  
        n.setNext(null);  
        n.setPrev(null);  
        n.setContainer(null);  
        return nElem;  
    }
```


The Sequence ADT in Java

```
public interface PositionalSequence extends  
PositionalContainer {
```

```
/* ***** Accessor Methods  
***** */
```

```
public Position first()  
throws EmptyContainerException;
```

```
public Position last()  
throws EmptyContainerException;
```

```
public Position before (Position p)  
throws InvalidPositionException,  
BoundaryViolationException;
```

```
public Position after (Position p)  
throws InvalidPositionException,  
BoundaryViolationException;
```

The Sequence ADT in Java (contd.)

```
/* ***** Information Methods  
***** */
```

```
public boolean isEmpty();
```

```
public boolean size();
```

```
public boolean isFirst (Position p)  
throws InvalidPositionException;
```

```
public boolean isLast (Position p)  
throws InvalidPositionException;
```

```
/* ***** Update Methods  
***** */
```

```
public Position insertFirst (Object element);
```

```
public Position insertLast (Object element);
```

The Sequence ADT in Java (contd.)

```
/* ***** More Update Methods  
***** */
```

```
public Position insertBefore (Position p, Object  
                             element)
```

```
throws InvalidPositionException;
```

```
public Position insertAfter (Position p, Object element)
```

```
throws InvalidPositionException;
```

```
public Object remove (Position p)
```

```
throws InvalidPositionException;
```

```
public Object replace (Position p, Object element)
```

```
throws InvalidPositionException;
```

```
public void swap (Position p, Position q)
```

```
throws InvalidPositionException;
```

```
}
```

Comparison of Sequence Implementations

- Is `replaceElemAtRank` $O(1)$ in a list?????

| Operations | Array | List |
|------------------------------------|--------|--------|
| size, isEmpty | $O(1)$ | $O(1)$ |
| atRank, rankOf, elemAtRank | $O(1)$ | $O(n)$ |
| first, last | $O(1)$ | $O(1)$ |
| before, after | $O(1)$ | $O(1)$ |
| replace, replaceElemAtRank, swap | $O(1)$ | $O(1)$ |
| insertElemAtRank, removeElemAtRank | $O(n)$ | $O(n)$ |
| insertFirst, insertLast | $O(1)$ | $O(1)$ |
| insertAfter, insertBefore | $O(n)$ | $O(1)$ |
| remove | $O(n)$ | $O(1)$ |

Bubble Sort

- A Bubble Sort works by scanning through a sequence and swapping a given element with the next one if the former is smaller than the latter:

| Pass | Swaps | Sequence |
|------|---------------------|--------------------|
| | | (5, 7, 2, 6, 9, 3) |
| 1st | 7 ↔ 2, 7 ↔ 6, 9 ↔ 3 | (5, 2, 6, 7, 3, 9) |
| 2nd | 5 ↔ 2, 7 ↔ 3 | (2, 5, 6, 3, 7, 9) |
| 3rd | 6 ↔ 3 | (2, 5, 3, 6, 7, 9) |
| 4th | 5 ↔ 3 | (2, 3, 5, 6, 7, 9) |

- Here is implementation of a Bubble Sort for an Array-Based Sequence:

```
public void static bubbleSort1(IntegerSequence s) {  
    int n = s.size();  
    for (int i=0; i<n; i++) // i-th pass  
        for (int j=0; j<n-i; j++)  
            if (s.atRank(j).intValue() >  
                s.atRank(j+1).intValue())  
                swap(s.positionAtRank(j), s.positionAtRank(j+1));  
}
```

Bubble Sort (contd.)

- This Implementation is designed for a sequence based on a doubly linked list.

```
public void static bubbleSort2(IntegerSequence s) {  
    int n = s.size();  
    IntegerSequencePosition prec, succ;  
  
    for (int i=0; i<n; i++) { // i-th pass  
        prec = s.firstPosition();  
        for (int j=0; j<n-i; j++) {  
            succ = s.after(prec);  
            if (prec.element().intValue() >  
succ.element().intValue())  
                swap(prec,succ);  
            else  
                prec = succ;  
        }  
    }  
}
```