Sets
Set Operations

- We represent a set by the sorted sequence of its elements.
- By specializing the auxiliary methods, the generic merge algorithm can be used to perform basic set operations:
  - union
  - intersection
  - subtraction
- The running time of an operation on sets $A$ and $B$ should be at most $O(n_A + n_B)$.

Set union:
- $aIsLess(a, S)$
  - $S.insertFirst(a)$
- $bIsLess(b, S)$
  - $S.insertLast(b)$
- $bothAreEqual(a, b, S)$
  - $S.insertLast(a)$

Set intersection:
- $aIsLess(a, S)$
  - \{ do nothing \}
- $bIsLess(b, S)$
  - \{ do nothing \}
- $bothAreEqual(a, b, S)$
  - $S.insertLast(a)$
We can implement a set with a list.
Elements are stored sorted according to some canonical ordering.
The space used is $O(n)$. 

Nodes storing set elements in order
Generic Merging

- Generalized merge of two sorted lists $A$ and $B$
- Template method `genericMerge`
- Auxiliary methods
  - `aIsLess`
  - `bIsLess`
  - `bothAreEqual`
- Runs in $O(n_A + n_B)$ time provided the auxiliary methods run in $O(1)$ time

**Algorithm** `genericMerge(A, B)`

1. $S \leftarrow$ empty sequence
2. **while** $\neg A.isEmpty() \land \neg B.isEmpty()$
   - $a \leftarrow A.first().element(); \ b \leftarrow B.first().element()$
   - **if** $a < b$
     - `aIsLess(a, S); A.remove(A.first())`
   - **else if** $b < a$
     - `bIsLess(b, S); B.remove(B.first())`
   - **else** `{ $b = a$ }
     - `bothAreEqual(a, b, S); A.remove(A.first()); B.remove(B.first())`
3. **while** $\neg A.isEmpty()$
   - `aIsLess(a, S); A.remove(A.first())`
4. **while** $\neg B.isEmpty()$
   - `bIsLess(b, S); B.remove(B.first())`
5. **return** $S$
Any of the set operations can be implemented using a generic merge.

For example:
- For **intersection**: only copy elements that are duplicated in both lists.
- For **union**: copy every element from both lists except for the duplicates.

All methods run in linear time.