Set Operations (§ 10.6)

- 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:
- $a$ is less than $S$.
  - $S$.insertFirst($a$)
- $b$ is less than $S$.
  - $S$.insertLast($b$)
- Both are equal.
  - $S$.insertLast($a$)

Set intersection:
- $a$ is less than $S$.
  - Do nothing.
- $b$ is less than $S$.
  - Do nothing.
- Both are equal.
  - $S$.insertLast($a$)

Storing a Set in a List

- We can implement a set with a list.
- Elements are stored sorted according to some canonical ordering.
- The space used is $O(n)$.

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)`

```plaintext
S ← empty sequence
while ¬A.isEmpty() ∧ ¬B.isEmpty() do
  a ← A.first().element(); b ← B.first().element()
  if a < b
    S.insertFirst(a); A.remove(A.first())
  else if b < a
    S.insertLast(b); B.remove(B.first())
  else
    b ← a
    bothAreEqual(a, b, S)
    A.remove(A.first()); B.remove(B.first())
  end
while ¬A.isEmpty() do
  A.remove(A.first())
end
while ¬B.isEmpty() do
  B.remove(B.first())
end
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.