Sets

Definitions

- A **set** is an unordered collection of elements, without duplicates that typically supports efficient membership tests.
  - Elements of a set are like keys of a map, but without any auxiliary values.
- A **multiset** (also known as a **bag**) is a set-like container that allows duplicates.
- A **multimap** is similar to a traditional map, in that it associates values with keys; however, in a multimap the same key can be mapped to multiple values.
  - For example, the index of a book maps a given term to one or more locations at which the term occurs.

Set ADT

S.add(e):	Add element e to the set. This has no effect if the set already contains e.
S.discard(e):	Remove element e from the set, if present. This has no effect if the set does not contain e.
e in S:	Return True if the set contains element e. In Python, this is implemented with the special __contains__ method.
len(S):	Return the number of elements in set S. In Python, this is implemented with the special method __len__.
iter(S):	Generate an iteration of all elements of the set. In Python, this is implemented with the special method __iter__.
S.remove(e):	Remove element e from the set. If the set does not contain e, raise a KeyError.
S.pop():	Remove and return an arbitrary element from the set. If the set is empty, raise a KeyError.
S.clear():	Remove all elements from the set.

Boolean Set Operations

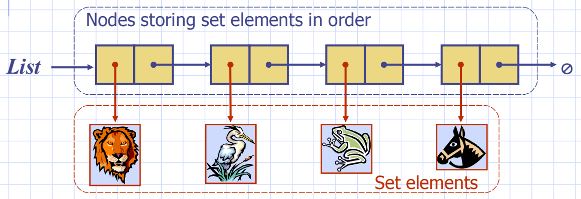
S == T:	Return True if sets S and T have identical contents.
S != T:	Return True if sets S and T are not equivalent.
S <= T:	Return True if set S is a subset of set T.
S < T:	Return True if set S is a *proper* subset of set T.
S >= T:	Return True if set S is a superset of set T.
S > T:	Return True if set S is a *proper* superset of set T.
S.isdisjoint(T):	Return True if sets S and T have no common elements.

Set Update Operations

S | T:	Return a new set representing the union of sets S and T.
S |= T:	Update set S to be the union of S and set T.
S & T:	Return a new set representing the intersection of sets S and T.
S &= T:	Update set S to be the intersection of S and set T.
S ^ T:	Return a new set representing the symmetric difference of sets S and T, that is, a set of elements that are in precisely one of S or T.
S ^= T:	Update set S to become the symmetric difference of itself and set T.
S − T:	Return a new set containing elements in S but not T.
S −= T:	Update set S to remove all common elements with set T.

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)
    S ← empty sequence
    while ¬A.isEmpty() ˄ ¬B.isEmpty()
        a ← A.first().element();  b ← 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())
    while ¬A.isEmpty()
        aIsLess(a, S);  A.remove(A.first())
    while ¬B.isEmpty()
        bIsLess(b, S);  B.remove(B.first())
    return S
```

Using Generic Merge for Set Operations

- Any of the set operations can be implemented using a generic merge
- For example:
  - For intersection: only copy elements that are duplicated in both list
  - For union: copy every element from both lists except for the duplicates
- All methods run in linear time