Generic Set Algorithms
Generic Set Algorithms

- Very useful software tools
- A part of the STL specification
- Implement set theory
  - Union
  - Intersection
  - Difference
  - Containment
  - Merge
- Runtime complexity in $O(\text{size})$
Set Algorithms
Assumptions and Outcomes

- **Assumptions**
  - Input ranges determined by input iterators
  - Output start determined by output iterator
  - Input ranges are sorted

- **Outcomes**
  - Output range is sorted
  - Output range is the set operation applied to the input ranges
```cpp
struct ltstr {
    bool operator()(const char* s1, const char* s2) const { return strcmp(s1, s2) < 0; }
};

int main() {
    const int N = 6;
    const char* a[N] = {"isomer", "ephemeral", "prosaic", "nugatory", "artichoke", "serif"};
    const char* b[N] = {"flat", "this", "artichoke", "frigate", "prosaic", "isomer"};

    set<const char*, ltstr> A(a, a + N);
    set<const char*, ltstr> B(b, b + N);
    set<const char*, ltstr> C;

    cout << "Set A: ";
    copy(A.begin(), A.end(),
         ostream_iterator<const char*>(cout, " "));
    cout << endl;

    cout << "Set B: ";
    copy(B.begin(), B.end(),
         ostream_iterator<const char*>(cout, " "));
    cout << endl;

    cout << "Union: ";
    set_union(A.begin(), A.end(),
              B.begin(), B.end(),
              ostream_iterator<const char*>(cout, " "), ltstr());
    cout << endl;

    cout << "Intersection: ";
    set_intersection(A.begin(), A.end(),
                     B.begin(), B.end(),
                     ostream_iterator<const char*>(cout, " "), ltstr());
    cout << endl;

    set_difference(A.begin(), A.end(), B.begin(), B.end(),
                   inserter(C, C.begin()), ltstr());

    cout << "Set C (difference of A and B): ";
    copy(C.begin(), C.end(),
         ostream_iterator<const char*>(cout, " "));
    cout << endl;
}
```
Set Algorithm Complexity

- **Unsorted input ranges**
  - $O(size^2)$ to iterate through range for each element
  - \{g, c, m, y, x, h, a\} union \{h, w, a, b\}

- **Sorted input ranges**
  - $O(size)$ to iterate through each range once
  - \{a, c, g, h, m, x, y\} union \{a, b, h, w\}
Sorted Range Control Structure

- Compare current elements from each input range
- Perform action based on comparison
- Increment past elements used for action
- Continue until an input range is exhausted
- Deal with tail of remaining range
Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{\}

B_1 \quad E_1

B_2 \quad E_2

D
Set Union Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a\}
Set Union Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a, b\}
Set Union Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a, b, c\}
Set Union Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Union Set: \( \{a, b, c, d\} \)
Set Union Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Union Set: \( \{a, b, c, d\} \)
Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a, b, c, d, g\}
Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a, b, c, d, g, m\}
Set Union Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Union Set: \{a, b, c, d, g, m\}
template <class I1, class I2, class I3>
void g_set_union(I1 B1, I1 E1, I2 B2, I2 E2, I3 D) {
    for (; B1 != E1 && B2 != E2; ++D) {
        if (*B1 < *B2) {
            *D = *B1;
            ++B1;
        } else if (*B2 < *B1) {
            *D = *B2;
            ++B2;
        } else {  // disallow duplicates
            *D = *B1;
            ++B1, ++B2;
        }
    }

    // Handle the tail
    while (B1 != E1) { *D++ = *B1++; }
    while (B2 != E2) { *D++ = *B2++; }
}
Set Merge Illustrated

Set 1: \( \{a, c, g, m\} \)

\[ B_1 \quad E_1 \]

Set 2: \( \{a, b, d\} \)

\[ B_2 \quad E_2 \]

Merged Set: \( \{\} \)

\[ D \]
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a\}
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a, a\}
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a, a, b\}
Set Merge Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Merged Set: \( \{a, a, b, c\} \)
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a, a, b, c, d\}
Set Merge Illustrated

Set 1: \( \{a, c, g, m\} \)

\[ B_1 \quad E_1 \]

Set 2: \( \{a, b, d\} \)

\[ E_2 \quad B_2 \]

Merged Set: \( \{a, a, b, c, d\} \)

\[ D \]
Set Merge Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Merged Set: \( \{a, a, b, c, d, g\} \)
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a, a, b, c, d, g, m\}
Set Merge Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Merged Set: \{a, a, b, c, d, g, m\}
template <class I1, class I2, class I3>
void g_set_merge(I1 B1, I1 E1, I2 B2, I2 E2, I3 D) {
    while (B1 != E1 && B2 != E2) {
        if (*B2 < *B1) {
            *D++ = *B2++;
        } else { // allow duplicates
            *D++ = *B1++;
        }
    }
    while (B1 != E1) { *D++ = *B1++; }
    while (B2 != E2) { *D++ = *B2++; }
}
Set Intersection Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Intersection Set: \{\}
Set Intersection Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Intersection Set: \{a\}
Set Intersection Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Intersection Set: \{a\}
Set Intersection Illustrated

Set 1: \( \{a, c, g, m\} \)

\[ \begin{array}{c}
\uparrow \\
B_1 & E_1
\end{array} \]

Set 2: \( \{a, b, d\} \)

\[ \begin{array}{c}
\uparrow \uparrow \\
B_2 & E_2
\end{array} \]

Intersection Set: \( \{a\} \)

\[ \begin{array}{c}
\uparrow \\
D
\end{array} \]
Set Intersection Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Intersection Set: \{a\}
Set Intersection Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Intersection Set: \{a\}
Set Intersection

template <class I1, class I2, class I3>
void g_set_intersection(I1 B1, I1 E1, I2 B2, I2 E2, I3 D) {
    while (B1 != E1 && B2 != E2) {
        if (*B2 < *B1) { // *B2 not in set 1
            ++B2;
        } else if (*B1 < *B2) { // *B1 not in set 2
            ++B1;
        } else {
            *D++ = *B1;
            ++B1;
            ++B2;
        }
    }
}
}
Set Difference Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Difference Set: \{\}
Set Difference Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Difference Set: \{\}

Set Difference Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Difference Set: \( \{\} \)
Set Difference Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)

Difference Set: \( \{c\} \)
Set Difference Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Difference Set: \{c\}
Set Difference Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Difference Set: \{c\}
Set Difference Illustrated

Set 1: \(\{a, c, g, m\}\)

Set 2: \(\{a, b, d\}\)

Difference Set: \(\{c, g\}\)
Set Difference Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Difference Set: \{c, g, m\}
template <class I1, class I2, class I3>
void g_set_difference(I1 B1, I1 E1, I2 B2, I2 E2, I3 D) {
    while (B1 != E1 && B2 != E2) {
        if (*B2 < *B1) { // *B2 not in set 1
            ++B2;
        } else if (*B1 < *B2) { // *B1 not in set 2
            *D++ = *B1++;
        } else {
            ++B1;
            ++B2;
        }
    }
    while (B1 != E1) { *D++ = *B1++; }
}
Set Containment Illustrated

Set 1: \( \{a, c, g, m\} \)

\[ \begin{array}{cccc}
B_1 & & & E_1 \\
& & & \\
\end{array} \]

Set 2: \( \{a, b, d\} \)

\[ \begin{array}{cccc}
B_2 & & & E_2 \\
& & & \\
\end{array} \]
Set Containment Illustrated

Set 1: \( \{a, c, g, m\} \)

Set 2: \( \{a, b, d\} \)
Set Containment Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}
Set Containment Illustrated

Set 1: \{a, c, g, m\}

Set 2: \{a, b, d\}

Return 0 (false)
template <class I1, class I2, class I3>
void g_subset_of(I1 B1, I1 E1, I2 B2, I2 E2) {
    while (B1 != E1 && B2 != E2) {
        if (*B1 < *B2) { // *B1 not in set 2
            return 0;
        } else if (*B2 < *B1) { // *B2 not in set 1
            ++B2;
        } else {
            ++B1;
            ++B2;
        }
    }
    if (B1 == E1) return 1;
    return 0;
}