1. Search Algorithms

- **`find`**
  Searches for the first occurrence of a value.

```
auto it = find(container.begin(), container.end(), value);
```

- **`find_if`**
  Searches for the first element that satisfies a predicate.

```
auto it = find_if(container.begin(), container.end(), predicate);
```

- **`find_if_not`**
  Searches for the first element that does not satisfy a predicate.

```
auto it = find_if_not(container.begin(), container.end(), predicate);
```

- **`binary_search`**
  Checks if a value exists in a sorted range.

```
bool found = binary_search(container.begin(), container.end(), value);
```

- **`lower_bound`**
  Returns the first position where the value can be inserted to maintain order.

```
auto it = lower_bound(container.begin(), container.end(), value);
```

- **`upper_bound`**
  Returns the first position where the value is greater than the given value.

```
auto it = upper_bound(container.begin(), container.end(), value);
```

- **`equal_range`**
  Returns the range of elements equivalent to the value.

```
auto range = equal_range(container.begin(), container.end(), value);
```

2. Sorting Algorithms

- `sort`
  Sorts a range of elements.

```
sort(container.begin(), container.end());
```

- `stable_sort`
  Sorts a range of elements while preserving the original order of equal elements.

```
stable_sort(container.begin(), container.end());
```

- `partial_sort`
  Partially sorts a range up to a given position.

```
partial_sort(container.begin(), middle, container.end());
```

- `nth_element`
  Rearranges elements such that the nth element is the element it would be in a fully sorted sequence.

```
nth_element(container.begin(), container.begin() + n, container.end());
```

3. Modification Algorithms

- `reverse`
  Reverses the order of elements in a range.

```
reverse(container.begin(), container.end());
```

- `rotate`
  Rotates elements within a range.

```
rotate(container.begin(), middle, container.end());
```

- `shuffle`
  Randomly shuffles the elements in a range.

```
shuffle(container.begin(), container.end(), rng);
```

- `replace`
  Replaces all occurrences of a value within a range.

```
replace(container.begin(), container.end(), old_value, new_value);
```

- `replace_if`
  Replaces elements that satisfy a predicate.

```
replace_if(container.begin(), container.end(), predicate, new_value);
```

- `fill`
  Fills a range with a given value.

```
fill(container.begin(), container.end(), value);
```

- `fill_n`
  Fills the first n elements of a range with a value.

```
fill_n(container.begin(), n, value);
```

- `transform`
  Applies a transformation function to each element.

```
transform(container.begin(), container.end(), destination.begin(), transformation);
```

4. Combination Algorithms

- `copy`
  Copies elements from one range to another.

```
copy(source.begin(), source.end(), destination.begin());
```

- `copy_if`
  Copies elements that satisfy a predicate.

```
copy_if(source.begin(), source.end(), destination.begin(), predicate);
```

- `move`
  Moves elements from one range to another.

```
move(source.begin(), source.end(), destination.begin());
```

- `move_if_noexcept`
  Moves elements if their type is noexcept movable.

```
move_if_noexcept(source.begin(), source.end(), destination.begin());
```

- `swap`
  Swaps the elements between two ranges or containers.

```
swap(a, b);
```

- `swap_ranges`
  Swaps elements between two ranges.

```
swap_ranges(container1.begin(), container1.end(), container2.begin());
```

- `replace_copy`
  Copies elements from one range to another, replacing values.

```
replace_copy(container.begin(), container.end(),
               destination.begin(), old_value, new_value);
```

CPP

- **`replace_copy_if`**

  Copies elements from one range to another, replacing those that satisfy a predicate.

```
replace_copy_if(container.begin(), container.end(),
                destination.begin(), predicate, new_value);
```

5. Set Algorithms

- `set_union`
  Computes the union of two sorted ranges.

```
set_union(range1.begin(), range1.end(), range2.begin(), range2.end(), result.begin());
```

- `set_intersection`
  Computes the intersection of two sorted ranges.

```
set_intersection(range1.begin(), range1.end(),
                                 range2.begin(), range2.end(), result.begin());
```

- `set_difference`
  Computes the difference between two sorted ranges.

```
set_difference(range1.begin(), range1.end(),
                                 range2.begin(), range2.end(), result.begin());
```

- `set_symmetric_difference`
  Computes the symmetric difference between two sorted ranges.

```
set_symmetric_difference(range1.begin(), range1.end(),
                                 range2.begin(), range2.end(), result.begin());
```

- `includes`
  Checks if the first sorted range includes the second sorted range.

```
bool result = includes(range1.begin(), range1.end(), range2.begin(), range2.end());
```

6. Numeric Algorithms

- **`accumulate`**
  Computes the sum of elements in a range.

```
auto sum = accumulate(container.begin(), container.end(), 0);
```

- **`inner_product`**
  Computes the dot product of two ranges.

```
auto dot_product = inner_product(range1.begin(), range1.end(), range2.begin(), 0);
```

- **`partial_sum`**
  Computes the partial sums of a range.

```
partial_sum(container.begin(), container.end(), result.begin());
```

- **`adjacent_difference`**
  Computes the differences between adjacent elements in a range.

```
adjacent_difference(container.begin(), container.end(), result.begin());
```

- **`max_element`**
  Finds the maximum element in a range.

```
auto max_it = max_element(container.begin(), container.end());
```

- **`min_element`**
  Finds the minimum element in a range.

```
auto min_it = min_element(container.begin(), container.end());
```

- **`minmax_element`**
  Finds the minimum and maximum elements in a range.

```
auto result = minmax_element(container.begin(), container.end());
```

7. General Utility Algorithms

- `all_of`
  Checks if all elements in a range satisfy a predicate.

```
bool all_true = all_of(container.begin(), container.end(), predicate);
```

- `any_of`
  Checks if any element in a range satisfies a predicate.

```
bool any_true = any_of(container.begin(), container.end(), predicate);
```

- `none_of`
  Checks if no elements in a range satisfy a predicate.

```
bool none_satisfied = none_of(container.begin(), container.end(), predicate);
```

- `for_each`
  Applies a function to each element in a range.

```
for_each(container.begin(), container.end(), func);
```

- `count`
  Counts the number of occurrences of a value in a range.

```
auto count = count(container.begin(), container.end(), value);
```

- `count_if`
  Counts the number of elements satisfying a predicate.

```
auto count = count_if(container.begin(), container.end(), predicate);
```

- `find_end`
  Searches for the last occurrence of a subsequence within a range.

```
auto it = find_end(container.begin(), container.end(),
                                        subsequence.begin(), subsequence.end());
```

- **`find_first_of`**
  Searches for the first occurrence of any element from a set of elements.

```cpp
auto it = find_first_of(container.begin(), container.end(), set.begin(), set.end());
```

- **`is_sorted`**
  Checks if the elements in a range are sorted.

```cpp
bool sorted = is_sorted(container.begin(), container.end());
```

- **`is_sorted_until`**
  Finds the first element in a range that is not sorted.

```cpp
auto it = is_sorted_until(container.begin(), container.end());
```

- **`lexicographical_compare`**
  Compares two ranges lexicographically.

```cpp
bool result = lexicographical_compare(container1.begin(), container1.end(),
                                      container2.begin(), container2.end());
```

8. Other Algorithms

- `generate`
  Generates a sequence of values by applying a function.

```
generate(container.begin(), container.end(), generator);
```

- `generate_n`
  Generates a sequence of n values by applying a function.

```
generate_n(container.begin(), n, generator);
```

- `mismatch`
  Finds the first pair of elements in two ranges that are not equal.

```
auto result = mismatch(container1.begin(), container1.end(), container2.begin());
```