Rangified version of lexicographical_compare_three_way

1 Motivation and Scope

This document adds the wording for `ranges::lexicographical_compare_three_way`

2 Design Decisions

— The Comp function is restricted to return one of the comparison categories, and nothing else. Therefore -

— There is no reason to restrict the relation between the compared ranges in any way (e.g. three_way_comparable_with).

— Functions built on top of `ranges::lexicographical_compare_three_way` may restrict their input parameters if required.

— Functions built on top of `ranges::lexicographical_compare_three_way` such as (the yet to be defined) `ranges::sort_three_way()` should benefit from the additional information that can be found in the return value of `ranges::lexicographical_compare_three_way`, and even use it to indicate the user that the function ended in a specific state. E.g. sort_three_way() may report that the resulted sorted range is sorted from smallest to largest (or largest to smallest), all element are equal or even that the given range is unsortable.

3 Proposed Wording

3.1 Add to [algorithm.syn]

```
template<class InputIterator1, class InputIterator2>
  constexpr auto
    lexicographical_compare_three_way(InputIterator1 b1, InputIterator1 e1,
                                      InputIterator2 b2, InputIterator2 e2);

template <typename T, typename... U>
concept same-as-any-of = (same_as<T, U> or ...); // exposition only

template<
    input_iterator I1,
    input_iterator I2,
    class Comp,
    class Proj1,
    class Proj2
>
using lexicographical-compare-three-way-result-t =
    invoke_result_t<
        Comp,
        typename projected<I1, Proj1>::value_type,
```

```cpp
        typename projected<I2, Proj2>::value_type
    >; // exposition-only

constexpr bool is-lexicographical-compare-three-way-result-ordering =
        same-as-any-of<
            lexicographical-compare-three-way-result-t<
                I1, I2, Comp, Proj1, Proj2
            >,
            strong_ordering, weak_ordering, partial_ordering>; //exposition-only

template<
    input_iterator I1, sentinel_for S1,
    input_iterator I2, sentinel_for S2,
    class Comp = compare_three_way,
    class Proj1 = identity,
    class Proj2 = identity
>
requires
    is-lexicographical-compare-three-way-result-ordering<
        I1, I2, Comp, Proj1, Proj2
    >
constexpr auto
    ranges::lexicographical_compare_three_way(
        I1 first1,
        S1 last1,
        I2 first2,
        S2 last2,
        Comp comp = {},
        Proj1 proj1 = {},
        Proj2 proj2 = {}
) -> common_comparison_category_t<
            decltype(
                comp(proj1(*first1), proj2(*first2))
            ),
            strong_ordering
    >;

template<
    ranges::input_range R1,
    ranges::input_range R2,
    class Comp = compare_three_way,
    class Proj1 = identity,
    class Proj2 = identity
>
requires
    is-lexicographical-compare-three-way-result-ordering<
        iterator_t<R1>, iterator_t<R2>, Comp, Proj1, Proj2
    >
constexpr auto
    ranges::lexicographical_compare_three_way(
        R1&& r1,
        R2&& r2,
        Comp comp = {},
        Proj1 proj1 = {},
```

```
        Proj2 proj2 = {}
    ) -> common_comparison_category_t<
            decltype(
                comp(proj1(ranges::begin(r1)), proj2(ranges::begin(r2)))
            ),
            strong_ordering
    >;
```

3.2 Add to §27.8.12 [alg.three.way]

```
template<class InputIterator1, class InputIterator2>
constexpr auto
  lexicographical_compare_three_way(InputIterator1 b1, InputIterator1 e1,
                      InputIterator2 b2, InputIterator2 e2);

template <typename T, typename... U>
concept same-as-any-of = (same_as<T, U> or ...); // exposition only

template<
    input_iterator I1,
    input_iterator I2,
    class Comp,
    class Proj1,
    class Proj2
>
using lexicographical-compare-three-way-result-t =
    invoke_result_t<
        Comp,
        typename projected<I1, Proj1>::value_type,
        typename projected<I2, Proj2>::value_type
    >; // exposition-only

constexpr bool is-lexicographical-compare-three-way-result-ordering =
        same-as-any-of<
            lexicographical-compare-three-way-result-t<
                I1, I2, Comp, Proj1, Proj2
            >,
            strong_ordering, weak_ordering, partial_ordering>; //exposition-only

template<
    input_iterator I1, sentinel_for S1,
    input_iterator I2, sentinel_for S2,
    class Comp = compare_three_way,
    class Proj1 = identity,
    class Proj2 = identity
>
requires
    is-lexicographical-compare-three-way-result-ordering<
        I1, I2, Comp, Proj1, Proj2
    >
constexpr auto
    ranges::lexicographical_compare_three_way(
        I1 first1,
        S1 last1,
        I2 first2,
```

```cpp
        S2 last2,
        Comp comp = {},
        Proj1 proj1 = {},
        Proj2 proj2 = {}
    ) -> common_comparison_category_t<
            decltype(
                comp(proj1(*first1), proj2(*first2))
            ),
            strong_ordering
    >;

    template<
        ranges::input_range R1,
        ranges::input_range R2,
        class Comp = compare_three_way,
        class Proj1 = identity,
        class Proj2 = identity
    >
    requires
        is-lexicographical-compare-three-way-result-ordering<
            iterator_t<R1>, iterator_t<R2>, Comp, Proj1, Proj2
        >
    constexpr auto
        ranges::lexicographical_compare_three_way(
            R1&& r1,
            R2&& r2,
            Comp comp = {},
            Proj1 proj1 = {},
            Proj2 proj2 = {}
    ) -> common_comparison_category_t<
            decltype(
                comp(proj1(*ranges::begin(r1)), proj2(*ranges::begin(r2)))
            ),
            strong_ordering
    >;
```

1   — Let N be the minimum integer between distance(first1,s1) and distance(first2,s2). Let E(n) be comp(proj1(*(*first1* + *n*)), *proj2(*(first2 + n))).

2   — Returns: E(i), where i is the smallest integer in [0, N) such that E(i) != 0 is true, or (distance(first1,s1)) <=> distance(first2, s2) if no such integer exists.

3   — Complexity: At most N applications of comp, porj1, porj2.

4 Acknowledgements