(0) Consider the following specification and implementation for a `lower_bound` function:

```c++
/** Find the first element in a sorted array
 * that is not less than a value.
 *
 * @param [in] a Array to search.
 * @param [in] n The number of elements of @a a to search.
 * @param [in] v Value to search for.
 * @return An index into array @a a or @a n.
 *
 * @pre 0 <= @a n <= Size of @a a.
 * @pre For all i,j with 0 <= i < j < n,
 * @a a[i] <= @a a[j].
 * @post For all i,j with 0 <= i < result <= j < n,
 * @a a[i] < @a v <= @a a[j].
 *
 * O(\log(n)) operations. Requires operator< for type T.
 */

template < typename T>
int lower_bound (const T* a, int n, const T& v) {
    int low = 0;
    int high = n;
    while (low < high) {
        int mid = low + (high-low) / 2;
        if (a[mid] < v)
            low = mid+1;
        else
            high = mid;
    }
    return low;
}
```

This specification is not *consistent*: it only requires `operator<` but uses `<=` and `>=` in its pre- and post-conditions.

More subtly, but also more importantly, it is also not *minimal*: there are alternative (weaker) specifications that would allow this `lower_bound` to be used by a *wider range of inputs* – inputs that the current specification prevents.

Provide a more consistent and minimal specification.
(1) In class, we abstracted `lower_bound` to take a comparator function object rather than always rely on `operator<`. In fact, it’s possible to define an interface to `lower_bound` that takes only a function object, rather than a comparator and an item to find. Write out a complete doxygen specification comment for this `lower_bound`. The implementation is not required. Pay special attention to the signature and requirements of the function object.

(2) Why might the C++ standard library have implemented the version with item and comparator, rather than the version with only a predicate?