UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Sorted arrays

Douglas Wilhelm Harder, M.Math., LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

Outline

- In this lesson, we will:
  - Describe a sorted array
  - Describe and implement an algorithm to determine if an array is sorted
  - Describe a consistent interface
  - Consider what happens with erroneous loop bounds

Sorted arrays

- A sorted array is an array where:
  - Each entry is greater than or equal to all previous entries
- An easier definition is:
  - Each entry, ignoring the first, is greater than or equal to the previous entry
- Put another way, if the entries of the array are

$$a_0, a_1, \ldots, a_{N-1},$$

then these entries are sorted if and only if

$$a_j \le a_k \textit{ whenever } j \le k$$

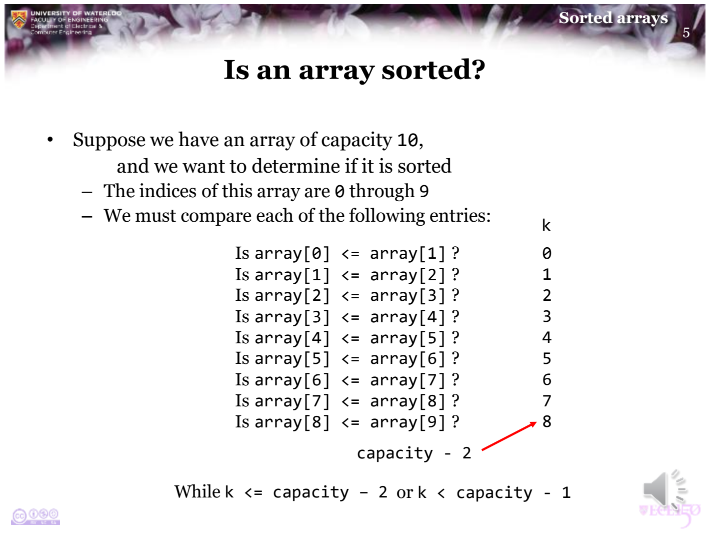
Is an array sorted?

- Only one entry need be out of order for the array to be considered to be not sorted:

  3.5, 7.8, 9.1, 10.7, 13.8, 15.7, 45.3, 83.6, 103.5, 199.2, $\boxed{187.3}$, 300.0

- Can we write a function to determine if an array is sorted?

```
bool is_sorted( double const array[],
                std::size_t const capacity );
```



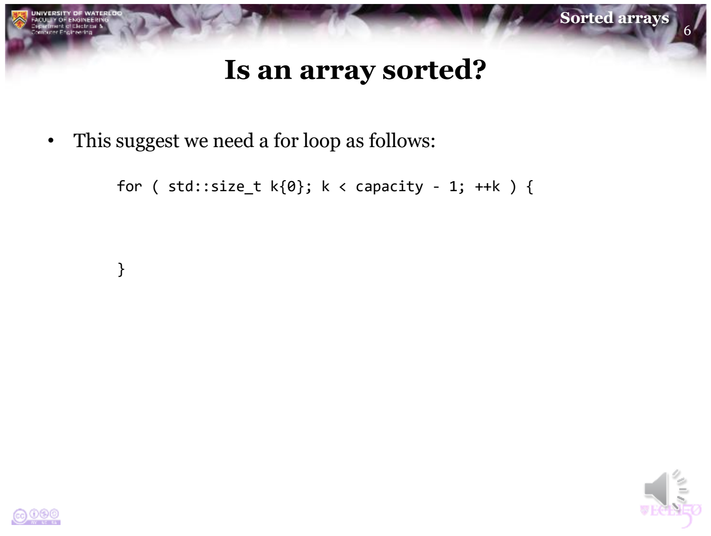
Is an array sorted?

- Suppose we have an array of capacity `10`,
  and we want to determine if it is sorted
  - The indices of this array are `0` through `9`
  - We must compare each of the following entries:

	k
Is `array[0] <= array[1]` ?	`0`
Is `array[1] <= array[2]` ?	`1`
Is `array[2] <= array[3]` ?	`2`
Is `array[3] <= array[4]` ?	`3`
Is `array[4] <= array[5]` ?	`4`
Is `array[5] <= array[6]` ?	`5`
Is `array[6] <= array[7]` ?	`6`
Is `array[7] <= array[8]` ?	`7`
Is `array[8] <= array[9]` ?	`8`

`capacity - 2`

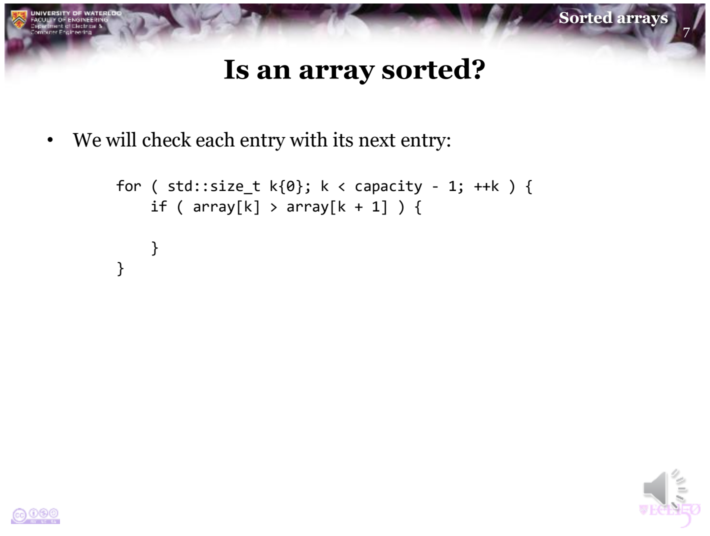
While `k <= capacity – 2` or `k < capacity - 1`



Is an array sorted?

- This suggest we need a for loop as follows:

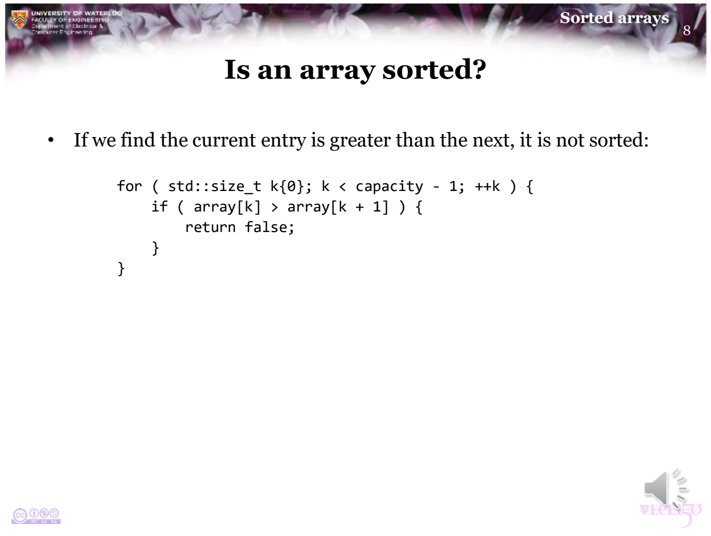
```
for ( std::size_t k{0}; k < capacity - 1; ++k ) {



}
```



Is an array sorted?

- We will check each entry with its next entry:

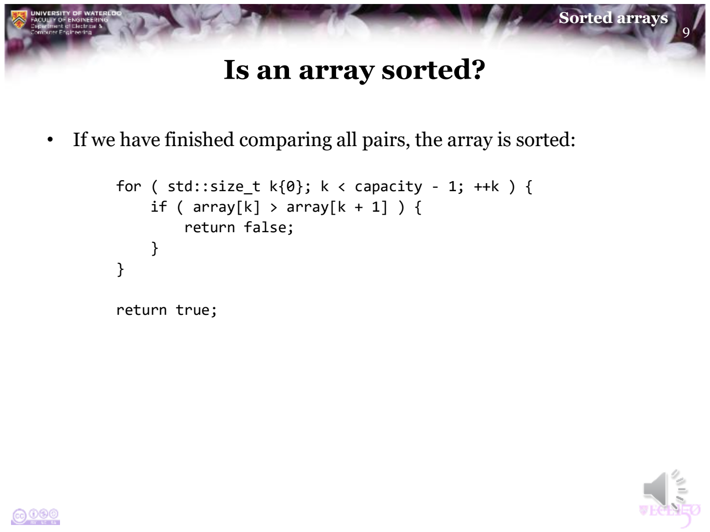
```
for ( std::size_t k{0}; k < capacity - 1; ++k ) {
    if ( array[k] > array[k + 1] ) {

    }
}
```



Is an array sorted?

- If we find the current entry is greater than the next, it is not sorted:

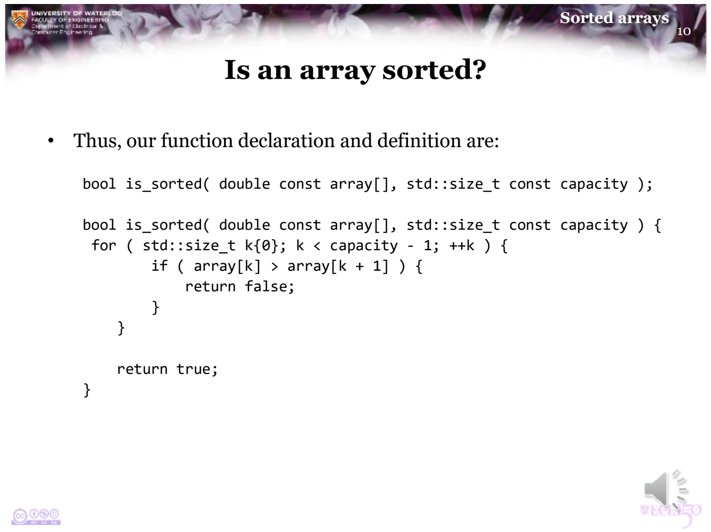
```
for ( std::size_t k{0}; k < capacity - 1; ++k ) {
    if ( array[k] > array[k + 1] ) {
        return false;
    }
}
```



Is an array sorted?

- If we have finished comparing all pairs, the array is sorted:

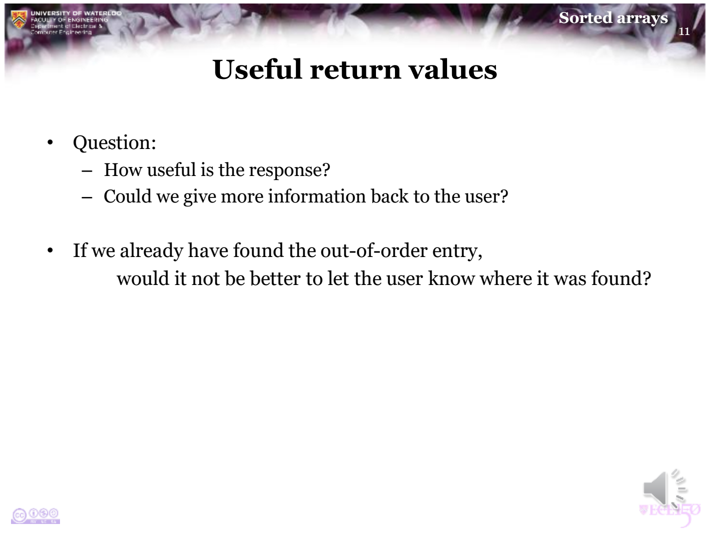
```
for ( std::size_t k{0}; k < capacity - 1; ++k ) {
    if ( array[k] > array[k + 1] ) {
        return false;
    }
}

return true;
```



Is an array sorted?

- Thus, our function declaration and definition are:

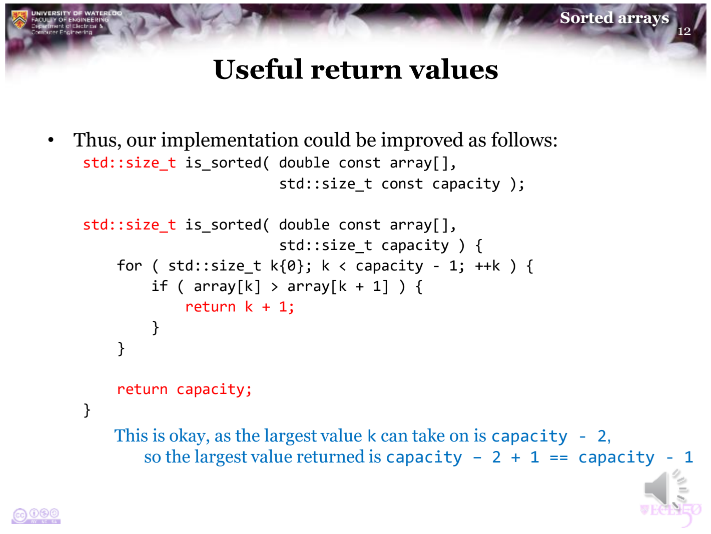
```
bool is_sorted( double const array[], std::size_t const capacity );

bool is_sorted( double const array[], std::size_t const capacity ) {
 for ( std::size_t k{0}; k < capacity - 1; ++k ) {
        if ( array[k] > array[k + 1] ) {
            return false;
        }
    }

    return true;
}
```



Useful return values

- Question:
  - How useful is the response?
  - Could we give more information back to the user?

- If we already have found the out-of-order entry,
  would it not be better to let the user know where it was found?

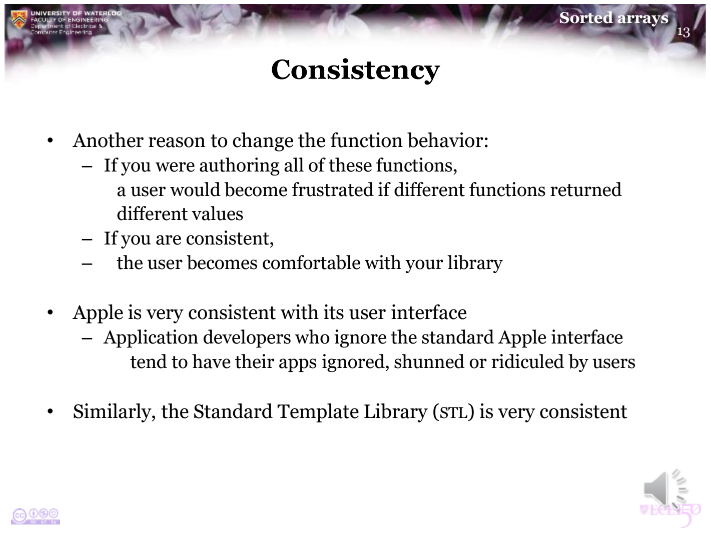


Useful return values

- Thus, our implementation could be improved as follows:

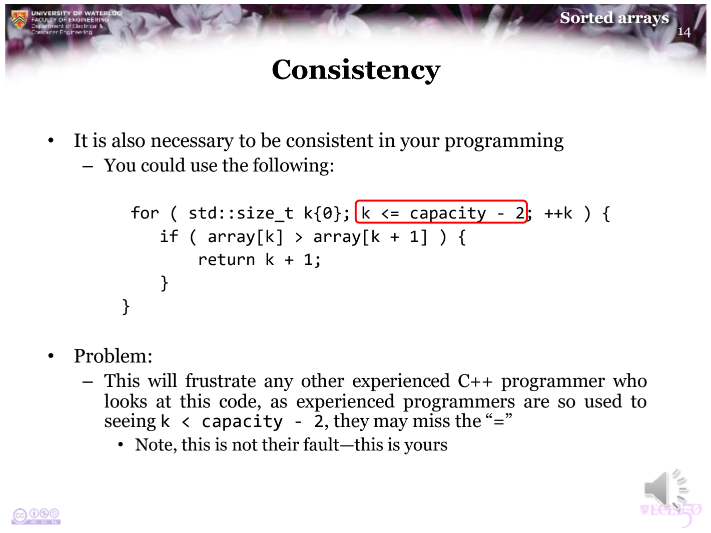
```
std::size_t is_sorted( double const array[],
                       std::size_t const capacity );

std::size_t is_sorted( double const array[],
                       std::size_t capacity ) {
    for ( std::size_t k{0}; k < capacity - 1; ++k ) {
        if ( array[k] > array[k + 1] ) {
            return k + 1;
        }
    }

    return capacity;
}
```

This is okay, as the largest value k can take on is `capacity - 2`,
so the largest value returned is `capacity - 2 + 1 == capacity - 1`



Consistency

- Another reason to change the function behavior:
  - If you were authoring all of these functions,
    a user would become frustrated if different functions returned different values
  - If you are consistent,
  - the user becomes comfortable with your library
- Apple is very consistent with its user interface
  - Application developers who ignore the standard Apple interface
    tend to have their apps ignored, shunned or ridiculed by users
- Similarly, the Standard Template Library (STL) is very consistent

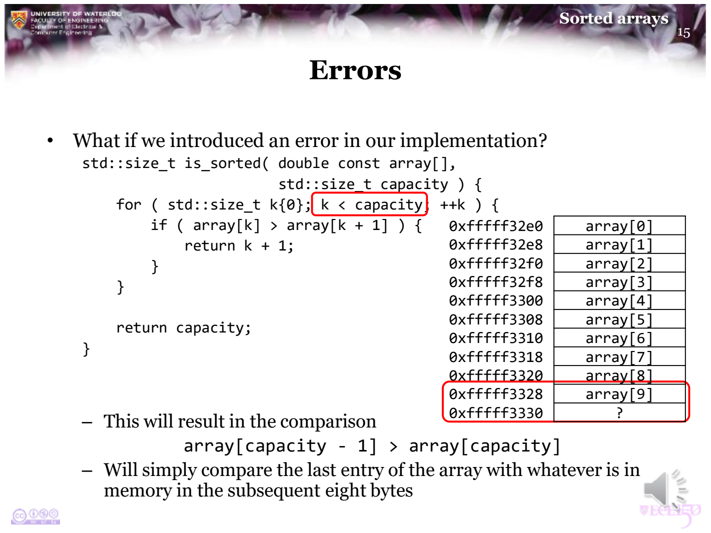


Consistency

- It is also necessary to be consistent in your programming
  - You could use the following:

```
for ( std::size_t k{0}; k <= capacity - 2; ++k ) {
    if ( array[k] > array[k + 1] ) {
        return k + 1;
    }
}
```

- Problem:
  - This will frustrate any other experienced C++ programmer who looks at this code, as experienced programmers are so used to seeing `k < capacity - 2`, they may miss the "="
    - Note, this is not their fault—this is yours



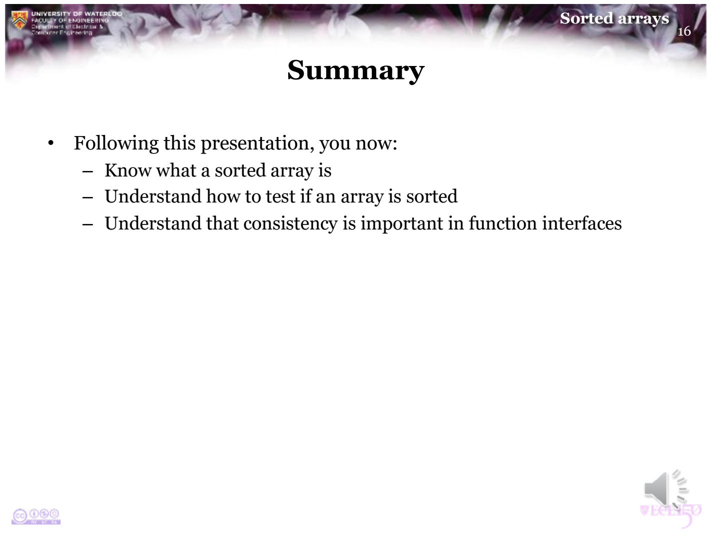
Errors

- What if we introduced an error in our implementation?

```
std::size_t is_sorted( double const array[],
                       std::size_t capacity ) {
    for ( std::size_t k{0}; k < capacity; ++k ) {
        if ( array[k] > array[k + 1] ) {
            return k + 1;
        }
    }

    return capacity;
}
```

0xfffff32e0	array[0]
0xfffff32e8	array[1]
0xfffff32f0	array[2]
0xfffff32f8	array[3]
0xfffff3300	array[4]
0xfffff3308	array[5]
0xfffff3310	array[6]
0xfffff3318	array[7]
0xfffff3320	array[8]
0xfffff3328	array[9]
0xfffff3330	?

  - This will result in the comparison
    `array[capacity - 1] > array[capacity]`
  - Will simply compare the last entry of the array with whatever is in memory in the subsequent eight bytes



Summary

- Following this presentation, you now:
  - Know what a sorted array is
  - Understand how to test if an array is sorted
  - Understand that consistency is important in function interfaces

References

[1] Wikipedia,
https://en.wikipedia.org/wiki/Sorted_array

[2] Dictionary of Algorithms and Data Structures (DADS)
https://xlinux.nist.gov/dads/HTML/sortedarray.html

Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer