Sorted arrays

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 $a_j \leq a_k$ whenever $j \leq k$

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, 187.3, 300.0

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

  ```cpp
  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:
    - \( \text{array}[0] \leq \text{array}[1] \)?
    - \( \text{array}[1] \leq \text{array}[2] \)?
    - \( \text{array}[2] \leq \text{array}[3] \)?
    - \( \text{array}[3] \leq \text{array}[4] \)?
    - \( \text{array}[4] \leq \text{array}[5] \)?
    - \( \text{array}[5] \leq \text{array}[6] \)?
    - \( \text{array}[6] \leq \text{array}[7] \)?
    - \( \text{array}[7] \leq \text{array}[8] \)?
    - \( \text{array}[8] \leq \text{array}[9] \)?

While \( k \leq \text{capacity} - 2 \) or \( k < \text{capacity} - 1 \)

Is an array sorted?

- We will check each entry with its next entry:

```cpp
for ( std::size_t k(0); k < capacity - 1; ++k ) {
    if ( array[k] > array[k + 1] ) {
        // Not sorted
    }
}
```

Is an array sorted?

- This suggest we need a for loop as follows:

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

Is an array sorted?

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

```cpp
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:

```cpp
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?

```cpp
std::size_t 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 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 \( k < \text{capacity} - 2 \), 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] ) { 
          0xffffffff32e0 array[0]
          0xffffffff32e0 array[1]
          0xffffffff32e0 array[2]
          0xffffffff32e0 array[3]
          0xffffffff3300 array[4]
          0xffffffff3300 array[5]
          0xffffffff3310 array[6]
          0xffffffff3310 array[7]
          0xffffffff3320 array[8]
          0xffffffff3328 array[9]
          0xffffffff3330 ?
      }
   }
   return capacity;
}
```

– 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


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

These slides are provided for the ECE 150 Fundamentals of Programming course taught at the University of Waterloo. The material in it reflects the authors’ best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.