CSCI 136: Data Structures and Advanced Programming
Lecture 13
Sorting, part 1
Instructor: Dan Barowy Williams

Topics
• More inheritance
• Comparison sorting

Your to-dos
1. Lab 4 (partner lab), due Tuesday 3/9 by 10pm.
2. Read before Wed: Bailey, Ch 6.4, 6.7-6.9.
Inheritance is a mechanism for defining a class in terms of another class. It is a labor-saving device employed to reduce code duplication. Inheritance allows programmers to specify a new implementation while:

1. maintaining the same behavior,
2. reusing code, and
3. extending the functionality of existing software.

Example

```java
public class Vector<E>
    extends AbstractList<E>
    implements Cloneable
```

Generic: any type of element
Borrows code from AbstractList
Behaves the same as Cloneable

(code)

Sorting algorithms
Sorting algorithm

A **sorting algorithm** is a **procedure** for transforming an unordered set of data into an ordered sequence.

A **comparison sorting algorithm** takes as input a set $S$ and a binary relation $<$ that defines an **ordering** on $S$.

Example order

**Example: lexicographical order** (aka, “dictionary order”):

Given two different sequences of the same length, $a_1a_2...a_k$ and $b_1b_2...b_k$, the first one is “less than” the second one for the lexicographical order, if $a_i < b_i$, for the first $i$ where $a_i$ and $b_i$ differ.

To compare sequences of different lengths, the shorter sequence is padded at the end with “blanks.”

Lexicographic order is a **total order**, meaning that there are **no ties**. A valid comparison sort only needs to be a **weak order** (i.e., ties are OK).

In-place sort

An **in-place sort** is a sort that takes an unordered set of elements as an array and **modifies** (“mutates”) the original array. Most in-place sort functions return **void**.

In principle, in-place sorts can be **faster** than out-of-place algorithms, since they **do not need to copy data**.

**Tradeoff**: make sure that you don’t need the original, unsorted data!

Bubble sort

**Bubble sort** is an **in-place sorting algorithm** in which the largest element “bubbles up” during each pass. Bubble sort makes $n-1$ passes through the data, performing pairwise comparisons of elements using $<$.

Bubble sort maintains the **invariant** (an always-true logical rule) that the rightmost $n\text{-numSorted}$ elements are sorted.

I.e., bubble sort builds a sorted order to the right.
Bubble sort: intuition

**Bubble sort** as a Hungarian dance.

Observe that two things are happening:
1. a comparison, and
2. a swap.


Bubble sort algorithm

```java
public static void bubbleSort(int data[], int n)
// pre: 0 <= n <= data.length
// post: values in data[0..n-1] in ascending order
{
    int numSorted = 0; // number of values in order
    int index; // general index
    while (numSorted < n)
    {
        // bubble a large element to higher array index
        for (index = 1; index < n-numSorted; index++)
        {
            if (data[index-1] > data[index])
                swap(data, index-1, index);
        }
        // at least one more value in place
        numSorted++;
    }
}
```

Recap & Next Class

**Today:**

- Inheritance
- Comparison sorting

**Next class:**

- More sorts