CSCI 136:
Data Structures
and
Advanced Programming

Lecture 15

Sorting, part 3

Instructor: Dan Barowy

Williams

Topics

- How do we sort data of any type? Comparators.
- Selection sort
- Insertion sort

Your to-dos

1. Lab 5 (solo lab), **due Tuesday 3/15 by 10pm**.
2. Reading: review (or catch up!) readings.

Quiz

What if…

… you wanted to sort **arbitrary objects**?

What's **problematic** with our bubble sort implementation?

(code)

Comparators

Comparators

We frequently have to sort data that is **more complex** than simple numbers.

For example, suppose we need to sort objects, like a `People[]`.

How do we define an order so that we can easily sort this?

`compare` to the rescue.

Comparator interface

The **`Comparator` interface** defines the method **`compare`** that lets us compare **two elements** of the same type.

```
public int compare(T o1, T o2)
```

Returns any **`int < 0`** when `o1` is "less than" `o2`.

Returns any **`int > 0`** when `o2` is "less than" `o1`.

Returns **`0`** otherwise.

(code)

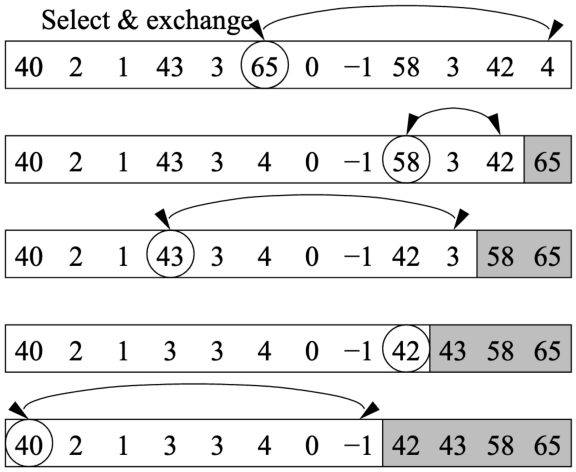
Selection sort

Selection sort is an **in-place sorting algorithm** in which the largest element is found during each pass. Selection sort makes **n-1** passes through the data, performing pairwise comparisons of elements using **<**. Unlike bubble sort, selection sort makes at most 1 swap during a pass.

Selection sort maintains the **invariant** that the rightmost **n-numUnsorted** elements are sorted.

I.e., selection sort builds a sorted order on the right.

Selection sort intuition



Selection sort

```
public static void selectionSort(int data[], int n)
// pre: 0 <= n <= data.length
// post: values in data[0..n-1] are in ascending order
{
    int numUnsorted = n;
    int index;      // general index
    int max;        // index of largest value
    while (numUnsorted > 0)
    {
        // determine maximum value in array
        max = 0;
        for (index = 1; index < numUnsorted; index++)
        {
            if (data[max] < data[index]) max = index;
        }
        swap(data,max,numUnsorted-1);
        numUnsorted--;
    }
}
```

Selection sort complexity

Selection sort is an **O(n^2)** sorting algorithm in the **worst case**. It is also **O(n^2)** in the **best case!**

Unlike other sorts, selection sort's runtime is **completely insensitive to the order of the data**.

Insertion sort

6 5 3 1 8 7 2 4

(see Wikipedia for animation)

Insertion sort

Insertion sort is a **sorting algorithm** in which the next element is **"inserted"** into a sorted array during each step. Insertion sort makes **n-1** passes through the sorted data, performing pairwise comparisons of elements using **<**.

Insertion sort maintains the **invariant** that the leftmost **n-numUnsorted** elements are sorted.

I.e., insertion sort builds a sorted order to the left.

Insertion sort complexity

Insertion sort is an **O(n^2)** sorting algorithm in the **worst case**. Insertion sort is **O(n)** in the best case.

Insertion sort algorithm

```
public static void insertionSort(int data[], int n)
// pre: 0 <= n <= data.length
// post: values in data[0..n-1] are in ascending order
{
    int numSorted = 1;      // number of values in place
    int index;              // general index
    while (numSorted < n)
    {
        // take the first unsorted value
        int temp = data[numSorted];
        // ...and insert it among the sorted:
        for (index = numSorted; index > 0; index--)
        {
            if (temp < data[index-1])
            {
                data[index] = data[index-1];
            } else {
                break;
            }
        }
        // reinsert value
        data[index] = temp;
        numSorted++;
    }
}
```

Recap & Next Class

Today:

- Comparators
- Selection sort
- Insertion sort

Next class:

- Fast sorts