CS 171: Introduction to Computer Science II

Simple Sorting (cont.) + Interface

Li Xiong
Today

• Simple sorting algorithms (cont.)
  – Bubble sort
  – Selection sort
  – Insertion sort

• Interface
**Sorting problem**

**Ex.** Student records in a university.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Value</th>
<th>Phone Number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
</tbody>
</table>

**Sort.** Rearrange array of \( N \) items into ascending order.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Value</th>
<th>Phone Number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
</tbody>
</table>
Two useful sorting abstractions

**Helper functions.** Refer to data through compares and exchanges.

**Less.** Is item \( v \) less than \( w \)?

```java
private static boolean less(Comparable v, Comparable w)
{
    return v.compareTo(w) < 0;
}
```

**Exchange.** Swap item in array \( a[] \) at index \( i \) with the one at index \( j \).

```java
private static void exch(Comparable[] a, int i, int j)
{
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```
Analysis of Bubble Sort

• Number of comparisons?

\[ \frac{N(N-1)}{2} = O(N^2) \]

• Number of swaps?

best case: \( O(1) \)

worst cast: \( \frac{N(N-1)}{2} = O(N^2) \)

average: \( \frac{N(N-1)}{4} = O(N^2) \)
Selection Sort

1. Keep track of the **index** of the smallest number in each round.
2. Swap the smallest number towards the beginning of the array.
3. Repeat the above two steps.
Selection Sort Implementation

```java
public class Selection {
    public static void sort(Comparable[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++) {
            int min = i;
            for (int j = i + 1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }

    private static boolean less(Comparable v, Comparable w) {
        return /* as before */;
    }

    private static void exch(Comparable[] a, int i, int j) {
        /* as before */
    }
}
```
Selection Sort

• Number of comparisons?
  \( O(N^2) \)

• Number of swaps?
  \( O(N) \)
Card Sorting Exercise

• How do you sort a hand of poker cards?
Insertion Sort

• Idea
  – Assume the left portion of the array is *partially sorted* (however, unlike selection sort, the elements are not necessarily in their final positions)
  – For each remaining element on the right portion, insert it to the left portion (similar to insertion in an ordered array).
  – Repeat until done.
Algorithm.  \( \uparrow \) scans from left to right.

Invariants.

- Entries to the left of \( \uparrow \) (including \( \uparrow \)) are in ascending order.
- Entries to the right of \( \uparrow \) have not yet been seen.
• Move the pointer to the right.

    i++;

• Moving from right to left, exchange \( a[i] \) with each larger entry to its left.

    for (int j = i; j > 0; j--)
        if (less(a[j], a[j-1]))
            exch(a, j, j-1);
        else break;
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0; j--)
                if (less(a[j], a[j-1]))
                    exch(a, j, j-1);
                else break;
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */ }
}
Insertion Sort

• Online demo
  – http://www.sorting-algorithms.com/insertion-sort

• Romanian dance demo
  – http://www.youtube.com/watch?v=ROaiU379l3U
Insertion Sort

• Number of comparisons?

• Number of exchanges?
Insertion sort

• Best case
  – N-1 comparisons
  – 0 exchanges

• Worst case
  – \( \sim N^2/2 \) comparisons
  – \( \sim N^2/2 \) exchanges

• Average case
  – \( \sim N^2/4 \) comparisons
  – \( \sim N^2/4 \) exchanges
Summary

• Both selection and insertion sort are comparison based.
• Both have an average comparison cost of \( O(N^2) \)

• Later we will learn several faster sorting algorithms, with a typical cost of \( O(N \log N) \)
Hw2

• Implement Bubble Sort
• Compare the runtime for bubble sort, selection sort, and insertion sort
Java’s Sorting Methods

• **Primitive Type Arrays:**
  
  ```java
  Arrays.sort(int[]);
  Arrays.sort(int[], int fromIdx, int toIdx);
  Arrays.sort(float[]);
  ```

  ......

  What sorting algorithm is used?
Java’s Sorting Methods

• **Object Type Arrays:**

```java
Arrays.sort(Object[]);
Arrays.sort(Object[], Comparator);
Arrays.sort(Object[], int fromIdx, int toIdx, Comparator);
```

......

• **Comparator** is used to define how to compare two objects (i.e. which is bigger / smaller).

```java
int compare(T o1, T o2)
boolean equals(Object obj)
```

What sorting algorithm is used?
Today

• Simple sorting algorithms (cont.)
  – Bubble sort
  – Selection sort
  – Insertion sort

• Interface
public class Insertion{
    
    public static void sort(Comparable[] a) {
    
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0; j--)
                if (less(a[j], a[j-1]))
                    exch(a, j, j-1);
            else break;
    
        }
    
    private static boolean less(Comparable v, Comparable w) {
        /* as before */
    }

    private static void exch(Comparable[] a, int i, int j) {
        /* as before */
    }
}
**Goal.** Sort any type of data.

**Ex 1.** Sort random real numbers in ascending order.

*seems artificial, but stay tuned for an application*

```java
public class Experiment {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        Double[] a = new Double[N];
        for (int i = 0; i < N; i++)
            a[i] = StdRandom.uniform();
        Insertion.sort(a);
        for (int i = 0; i < N; i++)
            StdOut.println(a[i]);
    }
}
```
Goal. Sort any type of data.

Ex 2. Sort strings from file in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = In.readStrings(args[0]);
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

```bash
% more words3.txt
bed bug dad yet zoo ... all bad yes
% java StringSorter words3.txt
all bad bed bug dad ... yes yet zoo
```
**Callback Mechanism: Interface**

**client**

```java
import java.io.File;
public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```

**object implementation**

```java
public class File
implements Comparable<File>
{
    ...
    public int compareTo(File b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}
```

**Comparable interface (built in to Java)**

```java
public interface Comparable<Item>
{
    public int compareTo(Item that);
}
```

**sort implementation**

```java
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

*key point: no dependence on File data type*
Abstract Classes and Interfaces

• Abstract class and abstract methods
• Interfaces
Superclasses and Subclasses

<table>
<thead>
<tr>
<th>GeometricObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>- color: String</td>
</tr>
<tr>
<td>- filled: boolean</td>
</tr>
<tr>
<td>- dateCreated: java.util.Date</td>
</tr>
<tr>
<td>+ GeometricObject()</td>
</tr>
<tr>
<td>+ getColor(): String</td>
</tr>
<tr>
<td>+ setColor(color: String): void</td>
</tr>
<tr>
<td>+ isFilled(): boolean</td>
</tr>
<tr>
<td>+ setFilled(filled: boolean): void</td>
</tr>
<tr>
<td>+ getDateCreated(): java.util.Date</td>
</tr>
<tr>
<td>+ toString(): String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- radius: double</td>
</tr>
<tr>
<td>+ Circle()</td>
</tr>
<tr>
<td>+ Circle(radius: double)</td>
</tr>
<tr>
<td>+ getRadius(): double</td>
</tr>
<tr>
<td>+ setRadius(radius: double): void</td>
</tr>
<tr>
<td>+ getArea(): double</td>
</tr>
<tr>
<td>+ getPerimeter(): double</td>
</tr>
<tr>
<td>+ getDiameter(): double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- width: double</td>
</tr>
<tr>
<td>- height: double</td>
</tr>
<tr>
<td>+ Rectangle()</td>
</tr>
<tr>
<td>+ Rectangle(width: double, height: double)</td>
</tr>
<tr>
<td>+ getWidth(): double</td>
</tr>
<tr>
<td>+ setWidth(width: double): void</td>
</tr>
<tr>
<td>+ getHeight(): double</td>
</tr>
<tr>
<td>+ setHeight(height: double): void</td>
</tr>
<tr>
<td>+ getArea(): double</td>
</tr>
<tr>
<td>+ getPerimeter(): double</td>
</tr>
<tr>
<td>+ getDiameter(): double</td>
</tr>
</tbody>
</table>
Abstract Classes and Abstract Methods

**GeometricObject**
- color: String
- filled: boolean
- dateCreated: java.util.Date

**Abstract class**

Methods `getArea` and `getPerimeter` are overridden in Circle and Rectangle. Overridden methods are generally omitted in the UML diagram for subclasses.

**Circle**
- radius: double

+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double

**Rectangle**
- width: double
- height: double

+Rectangle()
+Rectangle(width: double, height: double)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
abstract method in abstract class

public abstract void method();

• If a class contains abstract methods, it must be declared abstract

• If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be declared abstract
Instance cannot be created from abstract class

• An abstract class cannot be instantiated using the `new` operator
• You can still define its constructors, which are invoked in the constructors of its subclasses
• For instance, the constructors of `GeometricObject` are invoked in the `Circle` class and the `Rectangle` class.
superclass of abstract class may be concrete

• A subclass can be abstract even if its superclass is concrete
• For example, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract
abstract class as type

• You cannot create an instance from an abstract class using the `new` operator, but an abstract class can be used as a data type

```java
GeometricObject obj = new Circle(10);

GeometricObject[] geo = new GeometricObject[10];
```
Review questions

• Which of the following declares an abstract method in an abstract Java class?

A. public abstract method();
B. public abstract void method();
C. public void abstract Method();
D. public void method() {}
E. public abstract void method() {}
Review questions

Which of the following statements regarding abstract methods are true?

A. An abstract class can have instances created using the constructor of the abstract class.
B. An abstract class can be extended.
C. A subclass of a non-abstract superclass can be abstract.
D. An abstract class can be used as a data type.
Review questions

Suppose A is an abstract class, B is a concrete subclass of A, and both A and B have a default constructor. Which of the following is correct?

A a = new A();
A a = new B();
B b = new A();
B b = new B();
Interfaces

• What is an interface?
• Why is an interface useful?
• How do you define an interface?
• How do you use an interface?
What is an interface?
Why is an interface useful?

• An interface is a classlike construct that contains only constants and abstract methods

• In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify behavior for objects
  – Specify objects that are comparable, edible, cloneable using appropriate interfaces such as Comparable, Edible, and Cloneable

• A class that implements an interface need to implement all the abstract methods
  – define Orange and Chicken classes that implement Edible interface
Interface is a Special Class

• Like an abstract class, you cannot create an instance from an interface using the new operator
• You can create an instance from a class that implements an interface
• You can use an interface as a data type for a variable, as the result of casting, and so on.
Define an Interface

```java
public interface InterfaceName {
    constant declarations;
    method signatures;
}

public interface Edible {
    /** Describe how to eat */
    public abstract String howToEat();
}
```
Omitting Modifiers in Interfaces

— All data fields are public final static (constants) in an interface

— All methods are public abstract in an interface

```java
public interface T1 {
    public static final int K = 1;
    public abstract void p();
}

public interface T1 {
    int K = 1;
    void p();
}
```
The Comparable Interface

// This interface is defined in java.lang
package java.lang;

public interface Comparable {
    public int compareTo(Object o);
}
String and Date Classes

• Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects

```java
public class String extends Object implements Comparable {
    // class body omitted
}
```

```java
public class Date extends Object implements Comparable {
    // class body omitted
}
```
Declaring Classes to Implement Comparable

```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```
Callback Mechanism: Interface

```java
import java.io.File;
public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```

**Comparable interface (built in to Java)**

```java
public interface Comparable<Item>
{
    public int compareTo(Item that);
}
```

**object implementation**

```java
public class File
implements Comparable<File>
{
    ...
    public int compareTo(File b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}
```

**sort implementation**

```java
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
        else break;
}
```

key point: no dependence on File data type