

CS-259

Data Structures with Java

Bubble Sort

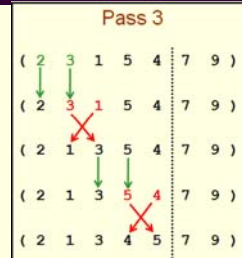
Instructor: **Joel Castellanos**

e-mail: joel@unm.edu

web: <http://cs.unm.edu/~joel/>

Course website:

<http://cs.unm.edu/~joel/cs259/>



1

Bubble Sort Algorithm

- Given a list of unsorted numbers.
- Each pass consists of:
 1. Starting with the first number in the list.
 2. Walking through the list and comparing each number with the number that is one position father down the list.
 3. In each comparison, if the first number is greater than the second, then swap the two numbers.
- Continue making passes until a full pass is completed without making any swaps.
- After n passes it is guaranteed that the last n numbers in the list will be sorted. Therefore, each pass need not compare the last n numbers.

-2-

Bubble Sort – Pass 1

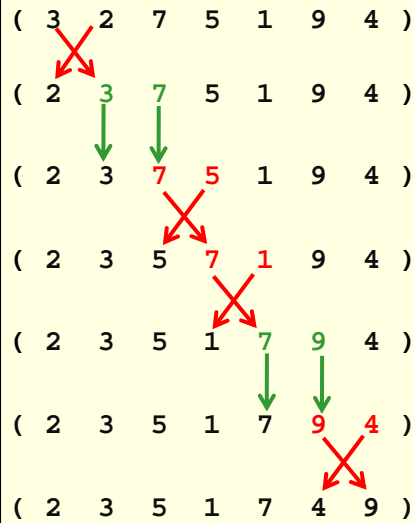
Original Numbers

{3, 2, 7, 5, 1, 9, 4}

Each pass consists of walking through the list and comparing each number with the number that is one position father down the list.

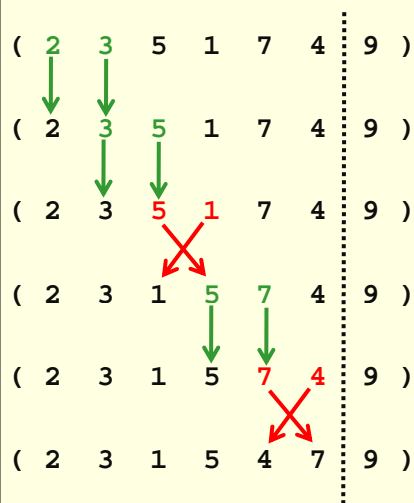
In each comparison, if the first number is greater than the second, then swap.

Pass 1



Bubble Sort – Pass 2

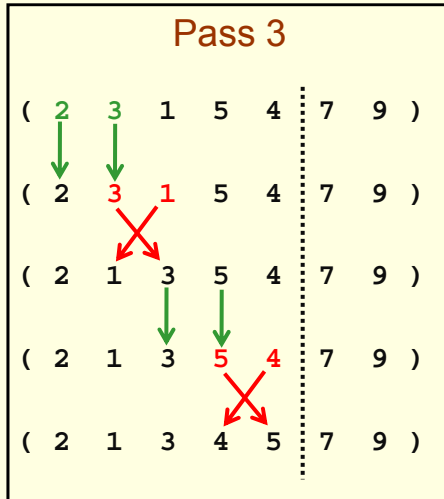
Pass 2



After the first pass, the largest number will have been moved to the end of the list.

Therefore, in pass 2, only the first 6 numbers need to be compared.

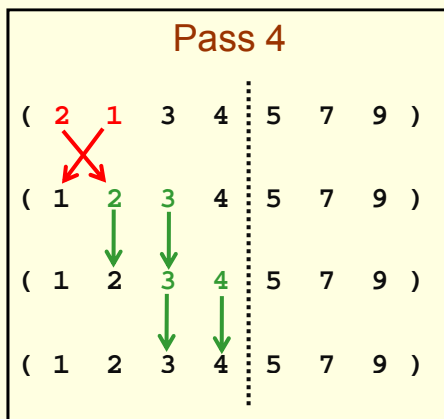
Bubble Sort – Pass 3



After the second pass, the second largest number will have been moved to the second to last position in the list.

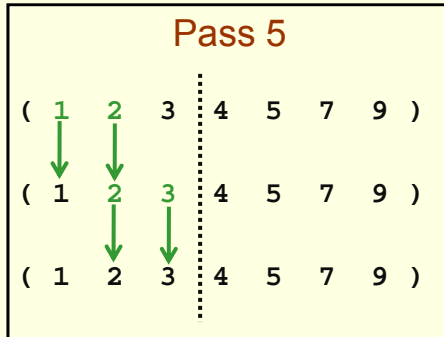
Therefore, in pass 3, only the first 5 numbers need to be compared.

Bubble Sort – Pass 4



After the first step of pass 4, the list is completely sorted; However, the algorithm does not “know” the list is sorted until a full pass is made with no swapping.

Bubble Sort – Pass 5



After 4 passes, the last 4 numbers are sorted, so only the first 3 numbers need to be compared.

Nothing is swapped in pass 5. Therefore, the list is fully sorted.

Bubble Sort: Performance

- Bubble sort has worst-case and average complexity both $O(n^2)$, where n is the number of items being sorted.
- There exist many sorting algorithms with the substantially better worst-case or average complexity of $O(n \log n)$.
- For example, for $n = 1$ million,
 - $n^2 = 1,000,000,000,000$
 - $n \log n = 13,815,511$
- Therefore bubble sort is not a practical sorting algorithm when n is large, except in rare specific applications where the array is known to be very close to being initially sorted.

Bubble Sort on an int Array

```
public static void bubbleSort(int[] array)
{ int length = array.length-1;
  boolean swap = true;

  while(swap)
  { swap = false;
    for (int i = 0; i < length; i++)
    {
      if (array[i] > array[i+1])
      { int tmp = array[i];
        array[i] = array[i+1];
        array[i+1] = tmp;
        swap = true;
      }
    }
  }
}
```

- This code gives the correct answer, but it is inefficient.
- What change will give the greatest speedup?



-9-

Quiz: Bubble Sort Speedup

```
1. public static void bubbleSort(int[] array)
2. { int length = array.length-1;
3.   boolean swap = true;
4.   while(swap)
5.   { swap = false;
6.     for (int i = 0; i < length; i++)
7.     { if (array[i] > array[i+1])
8.       { int tmp = array[i];
9.         array[i] = array[i+1];
10.        array[i+1] = tmp;
11.        swap = true;
12.      }
13.    }
14.  }
15. }
```

Where is the correct place to add the code:
length--; ?

- a) Between lines 5 and 6.
- b) Between lines 11 and 12.
- c) Between lines 12 and 13.
- d) Between lines 13 and 14.

-10-

Case for a static Method

```
public static void main(String[] args)
{
    int[] a = {1, 3, 4, 5, 2, 7, 0};
    int[] b = {8, 8, 2, 9, 4, 4};
    int[] c = {2, 1, 3, 9, 7, 6, 5, 8, 4};
    bubbleSort(a); bubbleSort(b); bubbleSort(c);
    for (int i=0; i<7; i++)
    { System.out.print(a[i]+" ");
    }
    System.out.println();
    for (int i=0; i<6; i++)
    { System.out.print(b[i]+" ");
    }
    System.out.println();
    for (int i=0; i<9; i++)
    { System.out.print(c[i]+" ");
    }
    System.out.println();
}
-11-
```

Current Output:

```
0, 1, 2, 3, 4, 5, 7,
2, 4, 4, 8, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Desired Output

```
{0, 1, 2, 3, 4, 5, 7}
{2, 4, 4, 8, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

printIntArray(int[] a)

```
public static void printIntArray(int[] a)
{
    System.out.print("{");
    for (int i=0; i<a.length-1; i++)
    { System.out.print(a[i]+" ");
    }
    System.out.println(a[a.length-1] + "}");
}

public static void main(String[] args)
{
    int[] a = {1, 3, 4, 5, 2, 7, 0 };
    int[] b = {8, 8, 2, 9, 4, 4 };
    int[] c = {2, 1, 3, 9, 7, 6, 5, 8, 4};
    bubbleSort(a); printIntArray(a);
    bubbleSort(b); printIntArray(b);
    bubbleSort(c); printIntArray(c);
}
-12-
```

```
{0, 1, 2, 3, 4, 5, 7}
{2, 4, 4, 8, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Employee Object for Bubble Sort

```
public class Employee
{ private String name;
  private double salary;

  public Employee(String name, double salary)
  { //Copy the String object, not just the pointer.
    this.name = new String(name);
    this.salary = salary;
  }

  //Override the Object toString() method.
  public String toString()
  { return name + "(" + salary + ")";
  }
}
```

-13-

Employee Object: Part 2

```
public class HelloWorld
{ public static void main(String[] args)
  { Employee[] x = new Employee[4];
    x[0] = new Employee("MacFree", 31500.0);
    x[1] = new Employee("Sam", 5023.52);
    x[2] = new Employee("Bea", 1200000.0);
    x[3] = new Employee("Macfay", 25000.0);
    bubbleSort(x);
    System.out.print("x=" + arrayToStr(x));
  }

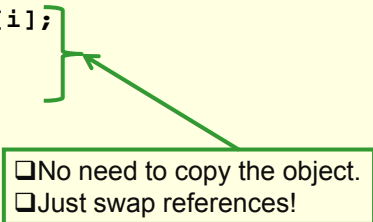
  public static String arrayToStr(Object[] array)
  { String str = "{";
    final String space = ", ";
    for (int i = 0; i<array.length-1; i++)
    { str = str + array[i] + space;
    }
    return str + array[array.length-1] + "}";
  }
}
```

Now, write bubbleSort(Employee[] emp)

-14-

Bubble Sort on a Record of Fields

```
public static void bubbleSort(Employee[] emp)
{ int length = emp.length-1;
  boolean swap = true;
  while(swap)
  { swap = false;
    for (int i=0; i < length; i++)
    { if (emp[i+1].name.compareToIgnoreCase(
        emp[i].name) < 0)
      {
        Employee tmp = emp[i];
        emp[i] = emp[i+1];
        emp[i+1] = tmp;
        swap = true;
      }
    }
    length--;
  }
}
```



- ❑ No need to copy the object.
- ❑ Just swap references!

-15-

Better Encapsulation for Sort Object

- bubbleSort() should not need to know what type of object it is sorting.
- thus,

```
public static void
    bubbleSort(Employee[] emp)
```

should be changed to


```
public static void
    bubbleSort(Comparable[] obj)
```

and `class Employee` needs to become:

```
class Employee implements
    Comparable<Employee>
```

-16-

Homework Results: 5 Point Quiz

- Hands off the computers! 
- When called on, tell me something about the work you did on: `bubbleSort(Comparable[] obj)`
 - What is Java's `Comparable`?
 - What are other parts of the interface?
 - What is a cool thing you learned about this topic?
 - What is a nontrivial problem you ran into?
 - By "nontrivial" computer scientists mean *interesting*.
 - "Could not find any information" is *not* interesting. ☹
- Do not tell me everything: get you 5 points and leave something for your classmates to say.

-17-

Interface Comparable <http://java.sun.com>

java.lang

Interface Comparable

- This interface imposes a total ordering on the objects of each class that implements it.
- A class that **implements** `Comparable` must implement a `compareTo` method.
- **public int compareTo(Object o)**
Compares **this** Object with the specified Object.

Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

-18-

bubbleSort of a Generic Type

```
public static void bubbleSort(Comparable[] obj)
{
    int length = obj.length-1;
    boolean swap = true;
    while(swap)
    {
        swap = false;
        for (int i=0; i < length; i++)
        {
            if (obj[i+1].compareTo(obj[i]) < 0)
            {
                Comparable tmp = obj[i];
                obj[i] = obj[i+1];
                obj[i+1] = tmp;
                swap = true;
            }
        }
        length--;
    }
}
```

The sorting method no longer needs to know the details of how its objects are compared.

- ❑ No need to copy the object.
- ❑ Just swap references!

-19-

Employee Implements compareTo()

```
public class Employee implements Comparable<Employee>
{
    private String name;
    private double salary;

    public Employee(String name, double salary)
    {
        this.name = new String(name);
        this.salary = salary;
    }

    public int compareTo(Employee other)
    {
        return name.compareToIgnoreCase(other.name);
    }

    public String toString()
    {
        return name + "(" + salary + ")";
    }
}
```

- Unlike bubbleSort(), the Employee class, does not just call compareTo(), it needs to implement it.
- The implementation of compareTo() needs to know that the object it is given contains: String name.

-22-

Access to Members of a Class

```
public class Employee implements Comparable<Employee>
{ private String name;
  private double salary;

  ...

  public int compareTo(Employee other)
  { return name.compareToIgnoreCase(other.name);
  }
}
```

- Note: The instance variable “name” is private. However, compareTo() is not only able to access its own instance of name, it is also able to access other.name.
- Methods outside the Employee class, however, would not be able to access this private field.

-23-

Put it Together: 5 Point Quiz

```
public class HelloWorld
{ public static void main(String[] args)
  { Employee[] x = new Employee[4];
    x[0] = new Employee("Mcsophia", 31500.0);
    x[1] = new Employee("Don", 5023.52);
    x[2] = new Employee("Qian", 1200000.0);
    x[3] = new Employee("McSylvain", 25000.0);

    //Must be bubbleSort(Comparable[] obj)
    bubbleSort(x);
    System.out.print("x=" + arrayToStr(x));
  }

  public static String arrayToStr(Object[] array)
  { ...
  }
}
```

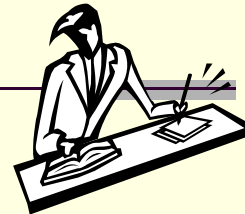
-24-

Lab: Extending Employee's compareTo()

1. Add a "behavior flag" that allows the user of the Employee class to choose between a *case sensitive* sort and a *case insensitive* sort:
 - Add Instance field:
`private boolean ignoreCase = true;`
 - Add and an `if` statement to `compareTo`
 - Add method:
`public void setIgnoreCase(boolean)`
2. Make `compareTo` sort by last name and, if a pair has the same last name, by first name.
 - Replace instance field `name` with two instance fields.
 - Update constructor: (`String nameFirst,`
`String, nameLast, double salary`)
 - Update the `main()` to use the new constructor.

-25-

Lab 6: Grading Rubric



- Total points: 20
- Deliverable: one file: `employee.java`
- Your instructor will use the API (Application Programmer's Interface) of your Employee class. The instructor's code will implement `main()` with 5 unknown test cases. You will receive 3 points for each correct result.
- Proper adherence to the CS-259 coding standard, including comments, is worth 5 points. You start with those 5 points and each error levies a -1 up to the maximum of 5 points.
- ~~2 points for each compiler warning.~~
- Due Date: Midnight on Thursday, Sept 17.

-26-