Comparable – an interface in Java

- We cannot create objects of an interface type
- We are allowed to create objects of a type that implements the interface, and then treat them as though they were an interface type

Bad:

```
Comparable c = new Comparable(); // error
```

Good:

```
Comparable c; // okay
MyClass m = new MyClass(); // MyClass implements Comparable
c = m; // legal – can manipulate c as though it is an object of type
            //Comparable, even though it is illegal to create such an
            // object
```

- Works not only with Comparable, but also with any interface
- can declare methods which take objects as parameters of type Comparable
- can declare sorting methods where the input is an array of type Comparable

Another wrinkle: It is legal to declare arrays of abstract or interface class type

The following is legal:

```
Comparable[] c = new Comparable[5];
```

- This does not create any new objects – it only creates space for the array.
- Within the array, each element of the array must be allocated to a new object.

```
MyClass[] m = new MyClass[5]; // this does not create any new
                              // objects – must initialize
                              //separately
for (int c = 0; c < 5; c++) { m[c] = new MyClass(); }
```

Bubble Sort

Review:
- start at the beginning of the list
- traverse the list until we find a member that is out of order
- move that element up the list until it is in order
- Continue from where we left off until the end of the list is reached.

Example

3 9 0 2 6
3 0 9 2 6
0 3 9 2 6
0 3 2 9 6
0 2 3 9 6
0 2 3 6 9

- Efficiency of bubble sort: Worst case scenario? List is already sorted in the wrong order. Number of sorting operations is

1+2+3+...+n
- Any such sum is O(n^2).

Divide-and-conquer sorting

- Divide-and-conquer sorts are much more efficient than bubble sort: O(n log n)

  1. Divide the array to be sorted into two sub arrays.
  2. Sort each sub-array
  3. Join the sub-arrays so that they are in order.

- note that “sort” is part of the sort procedure – recursive procedure.
- The method “sort” seems not to sort -- “dividing” and “joining” do all the sorting work.

- Merge sort: divide is trivial and join is complicated
- Quick sort: divide is complicated and join is trivial

Merge sort

- Divide: Divide the array to be sorted exactly in half.
- Join: Rearrange the elements of the two lists so that they are joined in order.

How do we join two sorted lists so that their elements are in order?

[1 5 6] [4 7 8]

- start at the beginning of both lists: 1, 4
- Pick the smallest element of these two: 1

1

- Move the pointer from 1 to 5: 5, 4
- Pick the smallest of these two: 4

1 4

- Move the pointer from 4 to 7: 5, 7
- Pick the smallest of these two: 5
- etc.

1 4 5 6 7 8

Example of merge sort:

4 2 0 9 1
Split: [4 2 0] [9 1]
    Sort 4 2 0
    Split: [4 2] [0]
    Sort 4 2
        Split: [4] [2]
        Sort 4 – only one element on this list, so sorted
        Sort 2 – sorted
        Join: 2 4
    Sort 0 – sorted
    Join [2 4] [0]: 0 2 4
Sort 9 1
    Split [9] [1]
    Sort 9 – already sorted
    Sort 1 – already sorted
    Join [9] [1] : 1 9
Join [0 2 4] [1 9]: 0 1 2 4 9 – sorted list.

- Interesting to remind ourselves that “sort” does nothing – the only thing “sort” does is determine when the list has length 1, in which case it is sorted by default.

Quick sort

- somewhat trickier than merge sort
- very widely used
- In this case, divide routine is complicated and the merge routine is simple.

1. Pick an arbitrary value of the list, called the “pivot”
2. Split the list into three sublists:
    1. Less than the pivot;
    2. Equal to the pivot;
    3. Greater than the pivot.
3. Perform the same procedure on the sublists.

- DO NOT follow the routine for QuickSort in the textbook – omits the “equal to the pivot” list.

4 2 0 9 1
Pivot: 4
Go through every element and put them in the three bins
Less than the pivot: 2 0 1
Equal to the pivot: 4
Greater than the pivot: 9

    2 0 1
    Pivot: 2
    Less than the pivot: 0 1

Equal to the pivot: 2
Greater than the pivot: empty

0 1
Pivot: 0
Less than the pivot: empty
Equal to the pivot: 0
Greater than the pivot: 1

empty : size 0, end sorting
0 : size 1, end sorting
1 : size 1, end sorting

Join: 0 1

2 : size 1, end sorting
empty : size 0, end sorting

Join: 0 1 2
4 : size 1, end sorting
9 : size 1, end sorting

Join: 0 1 2 4 9

Another example:

Sort 2 1 3

Split:
Pivot 2
Less than the pivot: 1
Equal to the pivot: 2

Greater than the pivot: 3

Sort the individual lists:

All of size 1, so no sorting can take place

Join them: 1 2 3

Project: Answers to questions

- I'm not looking for anything in particular, just something related to financial engineering.
- Difficulty: Assignments are worth 5%, and project is worth 30%. Per student, the project should be about 6 times as difficult as an assignment. Total difficulty: 6 x # of people.