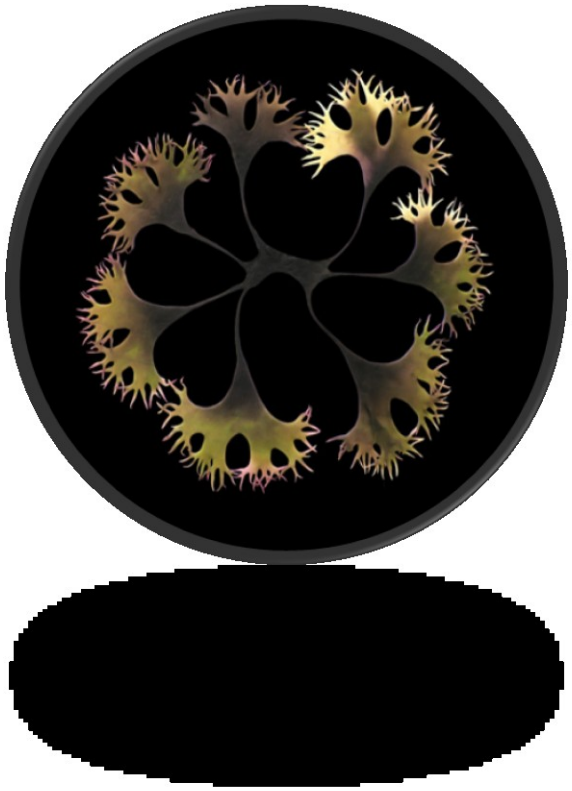


Sorting in Java

Data structures & Algorithms

MMUST

Sorting

:--Ok, so we have a list:

X1, X2, X3, X4, X5, X6, … XN

- We can sort this list either ascending or descending.
- What is ascending order??
- What do we mean by descending order?

Insertion sort

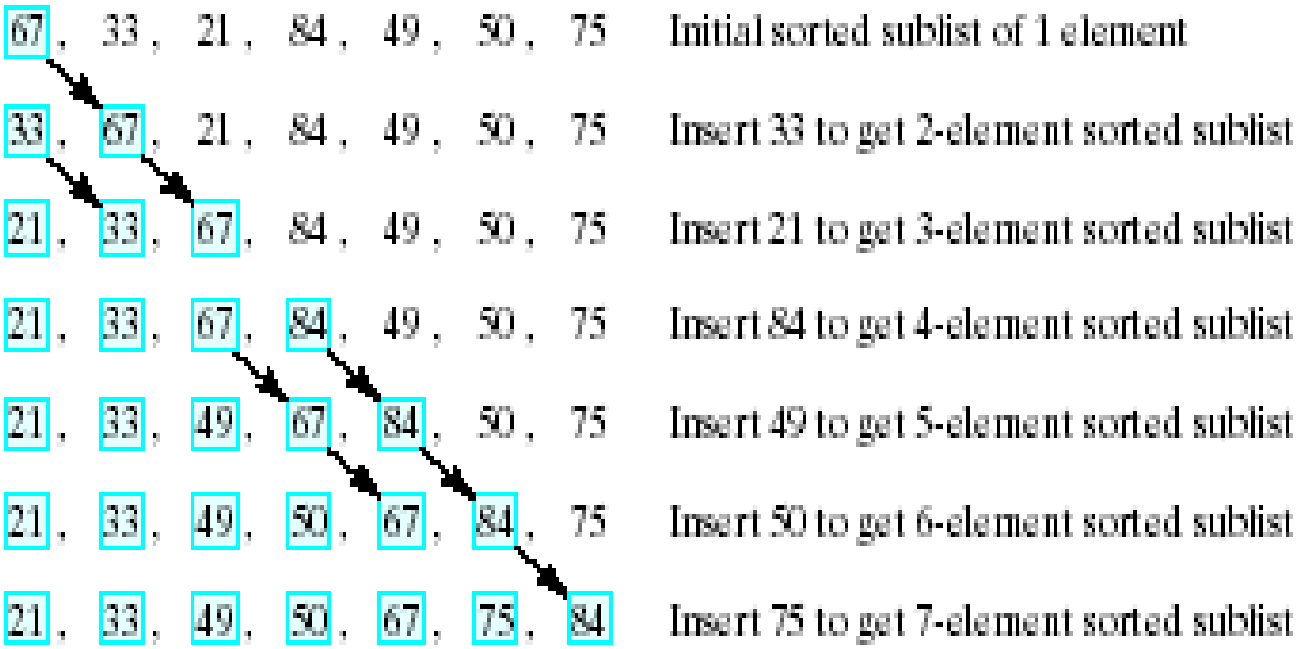
- In insertion sorting algorithm compare the value until all the prior elements are lesser than compared value is not found.
- This mean that the all previous values are lesser than compared value.
- Insertion sort is a good choice for small values and for nearly-sorted values.

Positive feature of insertion sorting

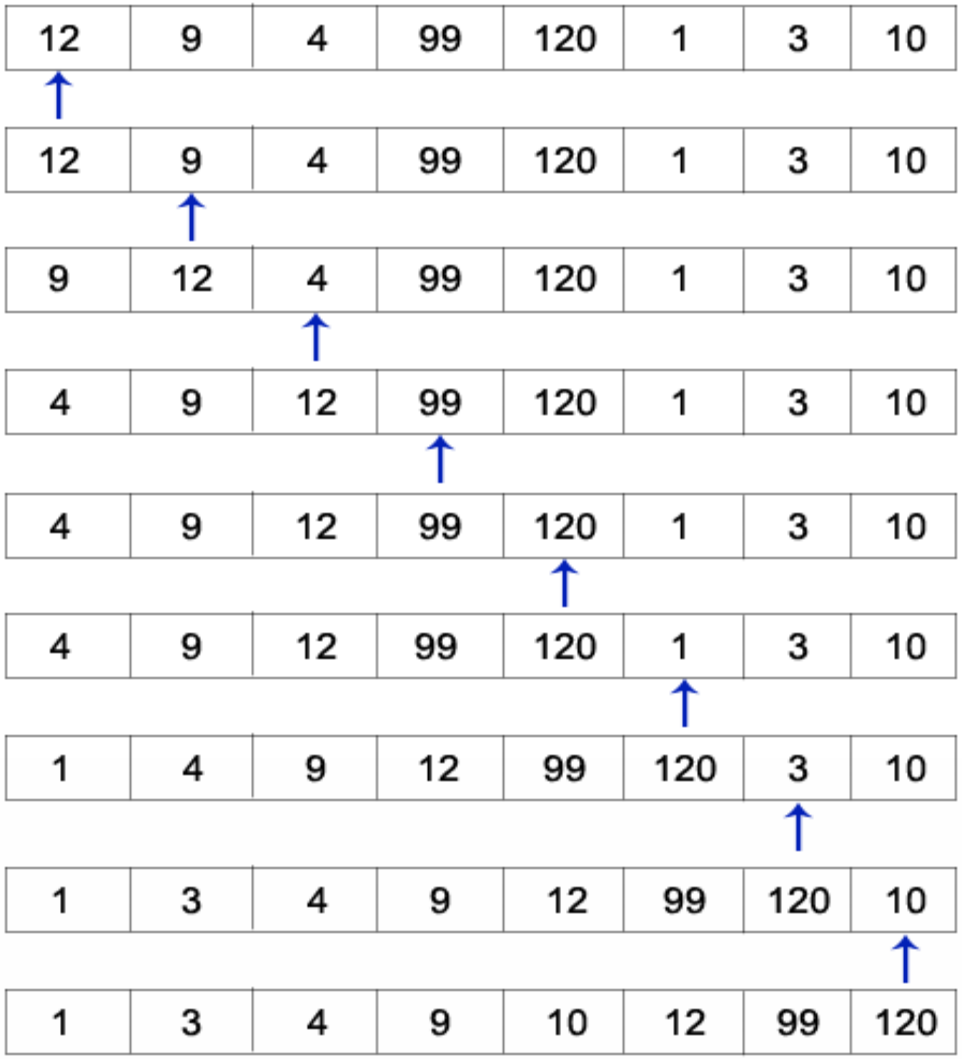
1. It is simple to implement
2. It is efficient on (quite) small data values
3. It is efficient on data sets which are already nearly sorted.

   There are two kinds of insertion sort:
   - Insert in-order into a new list
   - Reposition into our existing array

- The complexity of insertion sorting is O(n) at best case of an already sorted array and $O(n^2)$ at worst case



Cont. working with insertion sort



Sample code

```
public class InsertionSort{
    public static void main(String a[]){
      int i;
      int array[] = {12,9,4,99,120,1,3,10};
      System.out.println("\n\n       RoseIndia\n\n");
      System.out.println("       Selection Sort\n\n");
      System.out.println("Values Before the sort:\n");

      for(i = 0; i < array.length; i++)
      System.out.print( array[i]+"  ");
      System.out.println();
```

```
insertion_srt(array, array.length);
   System.out.print("Values after the sort:\n");

   for(i = 0; i <array.length; i++)
   System.out.print(array[i]+"  ");
   System.out.println();
   System.out.println("PAUSE");
 }
```

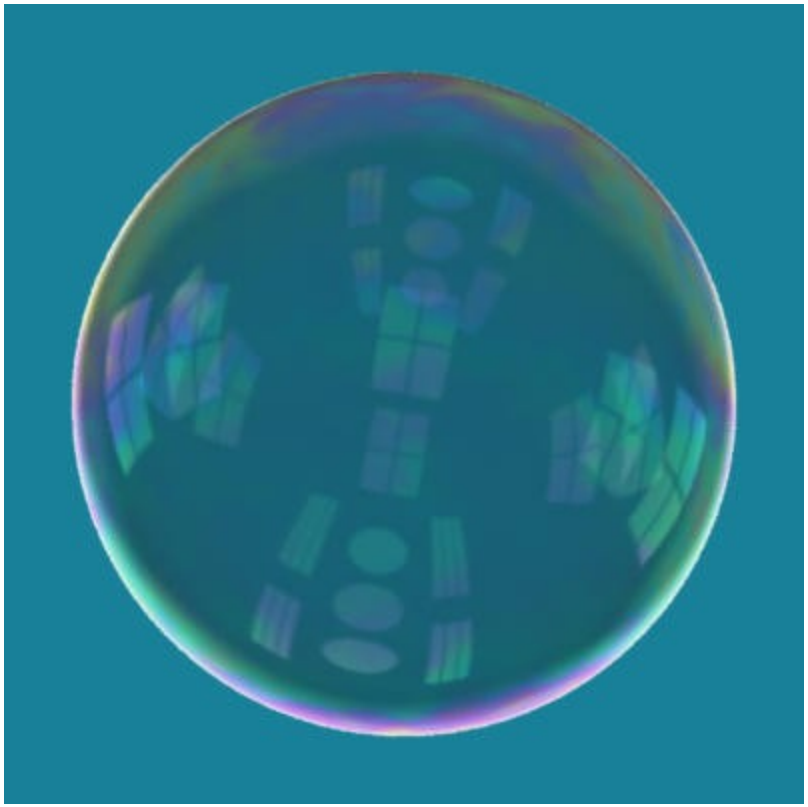
```
public static void insertion_srt(int array[], int n){
    for (int i = 1; i < n; i++){
      int j = i;
      int B = array[i];
      while ((j > 0) && (array[j-1] > B)){
        array[j] = array[j-1];
        j--;
      }
      array[j] = B;
    }
  }
}
```

Bubble Sort

- Bubble sort is a simple and well-known sorting algorithm
- Bubble sort belongs to $O(n^2)$ sorting algorithms, which makes it quite inefficient for sorting large data volumes
- Bubble sort is **stable** and **adaptive**.

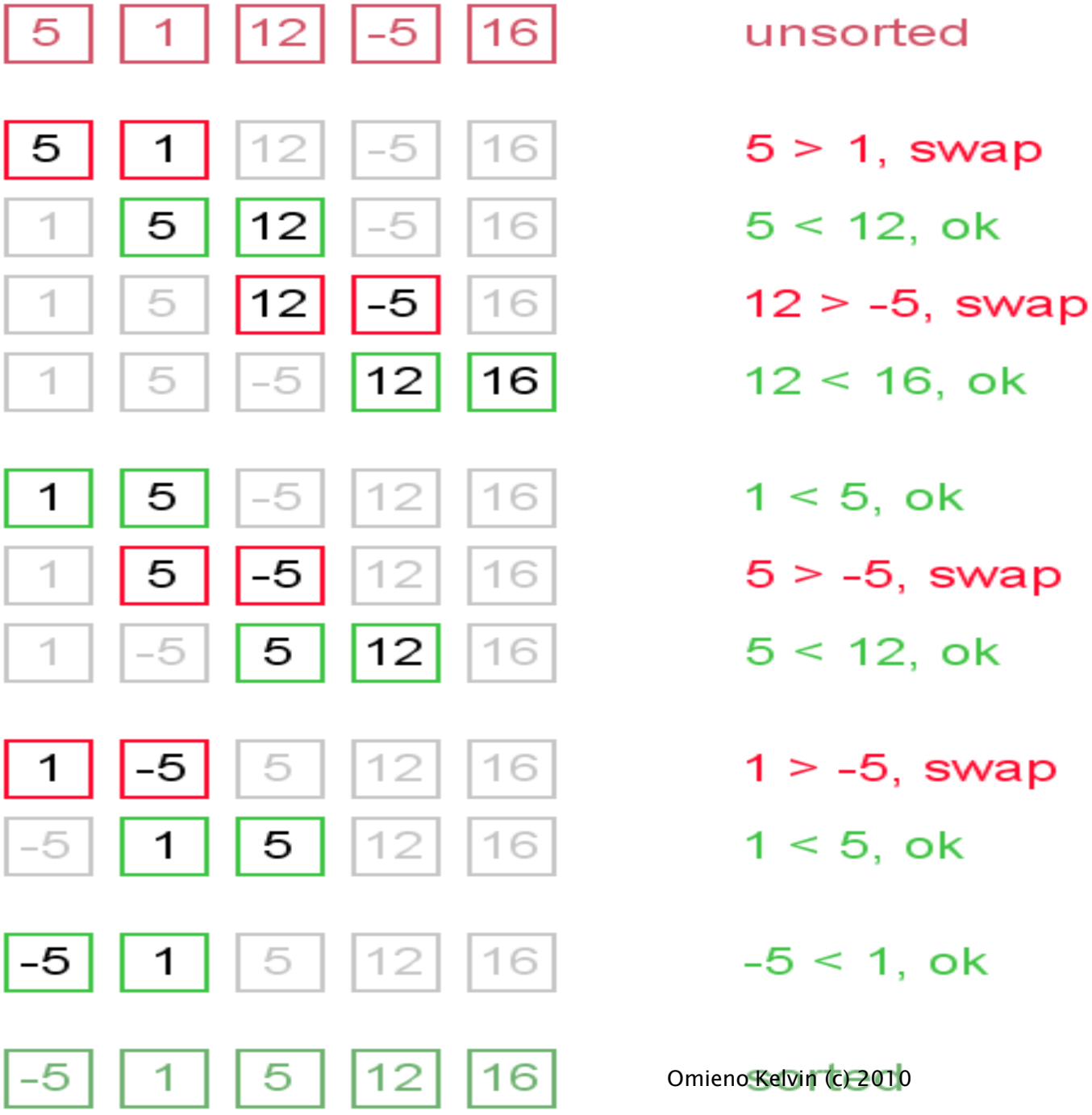
Cont.

- Has a good name.
- Easy to understand.
- Slow and crusty.
- What is its efficiency?



Algorithm

- Compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.
- If at least one swap has been done, repeat step 1.


5 1 12 -5 16
unsorted
5 1 12 -5 16
5 > 1, swap
1 5 12 -5 16
5 < 12, ok
1 5 12 -5 16
12 > -5, swap
1 5 -5 12 16
12 < 16, ok
1 5 -5 12 16
1 < 5, ok
1 5 -5 12 16
5 > -5, swap
1 -5 5 12 16
5 < 12, ok
1 -5 5 12 16
1 > -5, swap
-5 1 5 12 16
1 < 5, ok
-5 1 5 12 16
-5 < 1, ok
-5 1 5 12 16
sorted

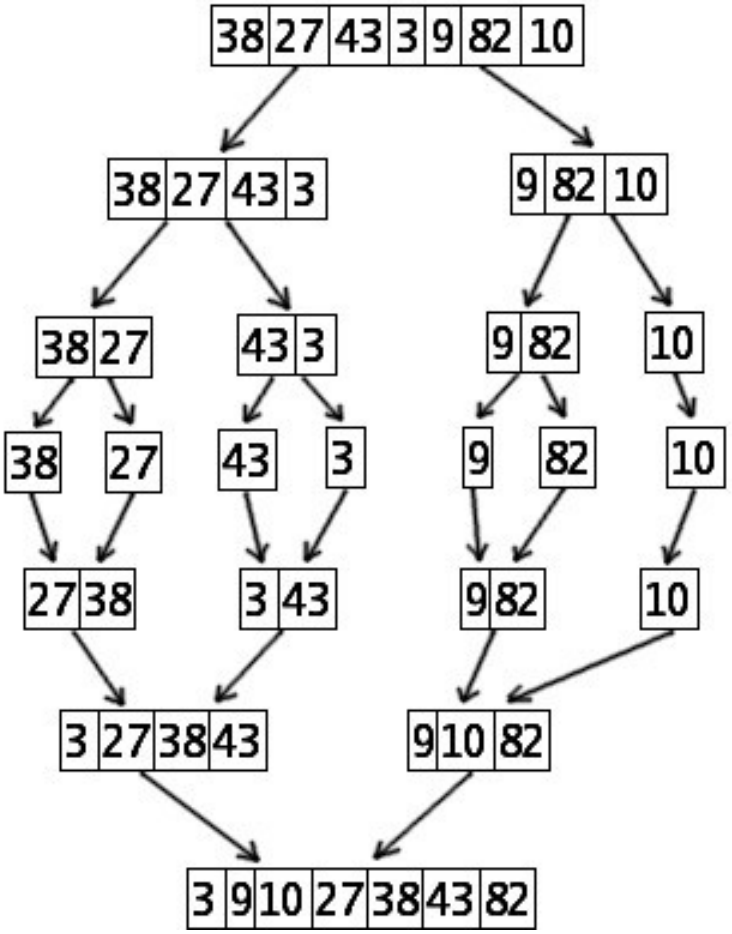
```
public void bubbleSort(int[] arr) {
     boolean swapped = true;
    int j = 0;
    int tmp;
   while (swapped) {
         swapped = false;
         j++;
```

Cont. code

```
for (int i = 0; i < arr.length - j; i++)
 {
                if (arr[i] > arr[i + 1]) {
                    tmp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = tmp;
                    swapped = true;
                }
            }
        }
}
```

Merge Sort

- Merge sort takes two (or more) inputs and sorts them!
- But how?
- By insertion!
- Uh oh… everyone has to get up now.



- It is a particularly good example of the divide and conquer algorithmic paradigm.
- Conceptually, merge sort works as follows:
  -Divide the unsorted list into two sub lists of about half the size
  - Sort each of the two sub lists
  - Merge the two sorted sub lists back into one sorted list.

Merge sort implementation

```
private static void merge( Comparable [ ] a, C
omparable [ ] tmpArray,  int leftPos, int rightP
os, int rightEnd ) {
        int leftEnd = rightPos - 1;
        int tmpPos = leftPos;
        int numElements = rightEnd -
 leftPos + 1;
```

```
// Main loop
        while( leftPos <= leftEnd && rightPos <= rightEnd )
            if( a[ leftPos ].compareTo( a[ rightPos ] ) <= 0 )
                tmpArray[ tmpPos++ ] = a[ leftPos++ ];
            else
                tmpArray[ tmpPos++ ] = a[ rightPos++ ];

        while( leftPos <= leftEnd )    // Copy rest of first half
            tmpArray[ tmpPos++ ] = a[ leftPos++ ];

        while( rightPos <= rightEnd )  // Copy rest of right ha
  lf
            tmpArray[ tmpPos++ ] = a[ rightPos++ ];

        // Copy tmpArray back
        for( int i = 0; i < numElements; i++, rightEnd-- )
            a[ rightEnd ] = tmpArray[ rightEnd ];
    }
```

Quick Sort

- The “Bad Boy” of sorts.
- This is recursive too.
- Again, everyone needs to get up.
- Hey…. These slides are pretty empty. But, you get to code this stuff!

- Read about this??????- Rapid sort??????????

Revision Questions

a) What is a spanning tree?

b) There are 8, 15, 13, 14 were there in four different trees. Which of them could have formed a full binary tree? Explain your answer.

c) Draw a diagram representing a LIFO structure for the program statement X=4+8*4/3

THANK YOU