COSC 121: Computer Programming II

Dr. Bowen Hui

University of British Columbia Okanagan

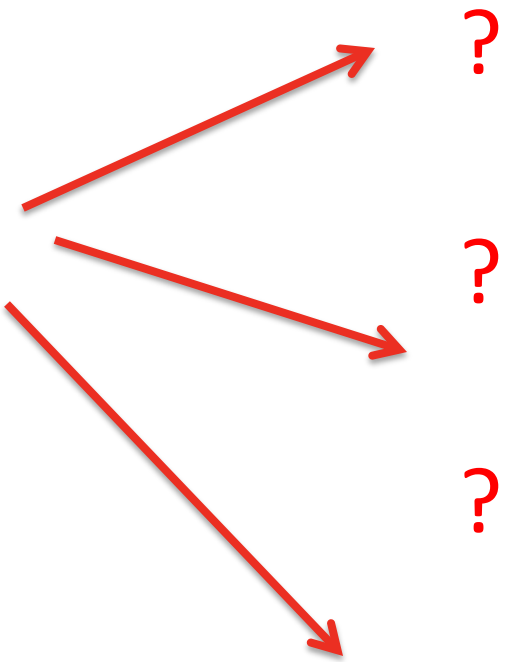
Sorting

- Sorting is the process of arranging a list of items in a particular order
  - Process is based on specific criteria
- Examples:
  - Sort test scores in ascending numeric order
  - Sort a list of people alphabetically by last name
- Why isn't there an obvious solution?

What to do with each paper?

Last Names on each paper:

Siemens
Leonenko
Momrath
Howard
Millard
Lam
Crowell
Schunk
Moore
Chan
Jin



Difficulties

- You don't know in advance:
  - All the information at once
    - See only one item at a time
  - How many items you're given
    - Each time you run the algorithm, the input is different
  - The distribution of the items
    - Everyone's name starts with "M"

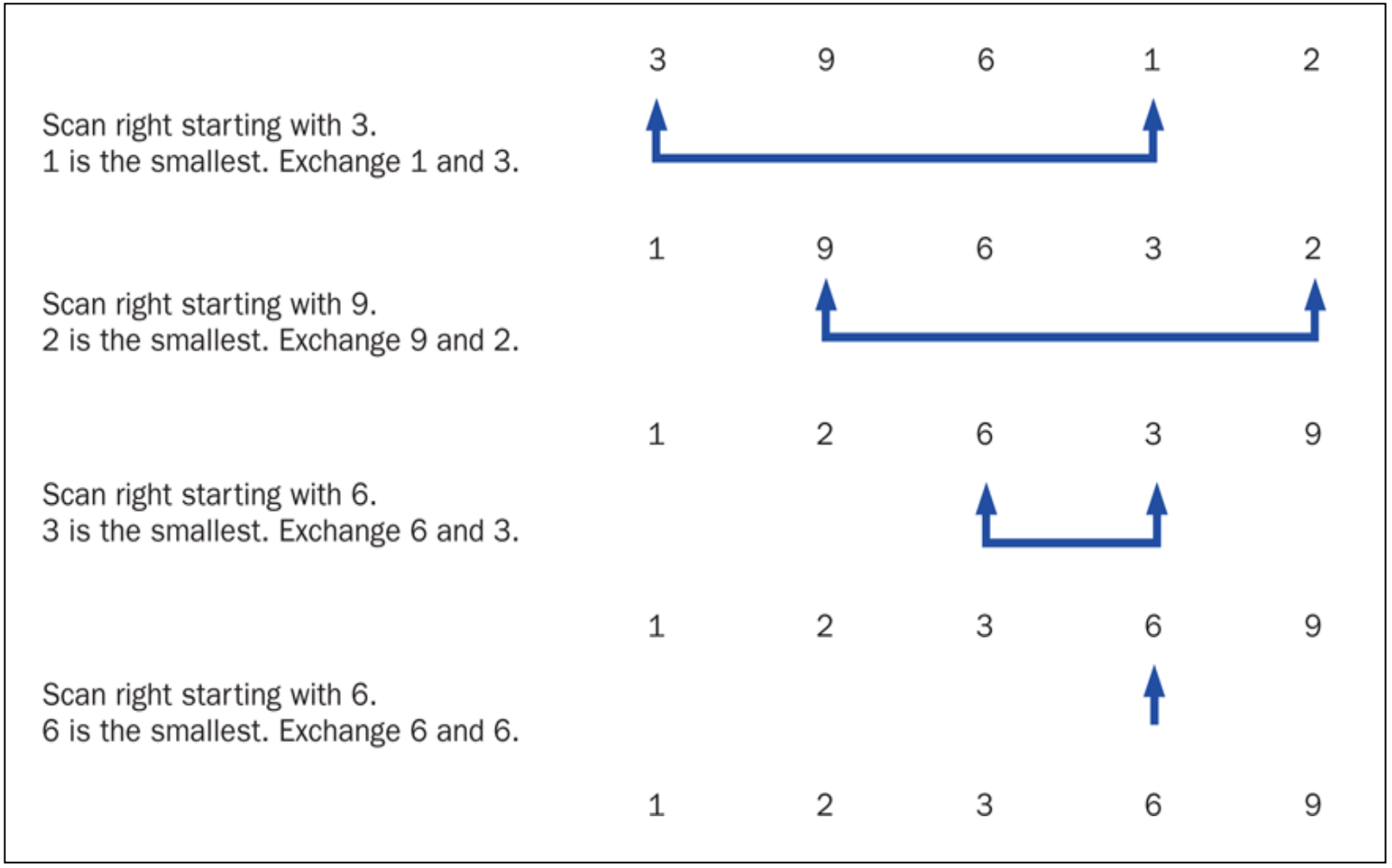
Sorting Algorithms

- Many algorithms exist varying in:
  - Time
  - Space (Memory)
- We will examine the following:
  - Selection Sort
  - Insertion Sort
  - Merge Sort (after Searching)
  - Quick Sort (after Searching)

Selection Sort

- The strategy of Selection Sort:
  - Find the smallest value in the list
  - Switch it with the value in the first position
  - Find the next smallest value in the list
  - Switch it with the value in the second position
  - Repeat until all values are in their proper places

Selection Sort Example



Swapping in Detail

- Selection sort includes the swapping of two values
- Requires three assignment statements and a temporary storage location
- Why won't this work?

```
first = second;
second = first;
```

Swapping in Detail

- Selection sort includes the swapping of two values
- Requires three assignment statements and a temporary storage location
- To swap the values of `first` and `second`:

```
temp = first;
first = second;
second = temp;
```

Defining Selection Sort

```
public class Sort
{
   public static void main (String[] args)
   {
      int[] arr = {5, 4, 7, 2, 9, 3, 1};

      System.out.println( "Original set of numbers … " );
      for( int i=0; i<arr.length; i++ )
        System.out.println( arr[i] );

      int[] rez = selectionSort( arr );

      System.out.println( ”Sorted set of numbers … " );
      for( i=0; i<rez.length; i++ )
        System.out.println( rez[i] );
   }
   public static int[] selectionSort( int[] nums ) { … }
}
```

Defining Selection Sort

```
public class Sort
{
    public static void main (String[] args)
    {
        int[] arr = {5, 4, 7, 2, 9, 3, 1};

        System.out.println( "Original set o
        for( int i=0; i<arr.length; i++ )
          System.out.println( arr[i] );

        int[] rez = selectionSort( arr );

        System.out.println( "Sorted set of
        for( i=0; i<rez.length; i++ )
          System.out.println( rez[i] );
    }
    public static int[] selectionSort( int
}
```

Output

```
Original set of numbers …
5
4
7
2
9
3
1
Sorted set of numbers …
1
2
3
4
5
7
9
```

Recall: Selection Sort

- Summary of strategy:
  - Find the smallest value in the list
  - Switch it with the value in the first position
  - Repeat “inwards”
- How to translate this into Java?

Recall: Selection Sort

- Summary of strategy:
  - Find the smallest value in the list
  - Switch it with the value in the first position
  - Repeat “inwards”
- How to translate this into Java?
  - Consider each value (left to right)
  - For each value, swap with smallest of the remaining values

Pseudocode

```
public static int[] selectionSort( int[] numbers )
{
  // 1. consider each number (from left to right)
  // 2. switch it with smallest value in remaining nums
  // a. find smallest value in remaining nums
  // b. swap two values
  // 3. return the sorted array
}
```

- How to convert each step into Java code?

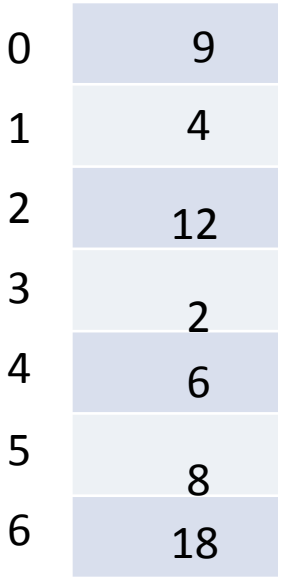
```
public static int[] selectionSort( int[] numbers )
{
   int i, j, min, temp;
   // 1. take each number (from left to right)
   for( i=0; i<numbers.length; i++ )
   {
      // 2. switch it with the smallest value in the remaining numbers
      // take the index of the i'th item
      min = i;

      //    a. find the smallest value in the remaining numbers
      // find smallest value in remaining list
      for( j=(i+1); j<numbers.length; j++ )
      {
        if( numbers[j] < numbers[min] )
          min = j;
      }

      //    b. swap two values
      // swap the values
      temp = numbers[min];
      numbers[min] = numbers[i];
      numbers[i]   = temp;
   }

   // 3. return the sorted array
   return numbers;
}
```

could use numbers.length-1

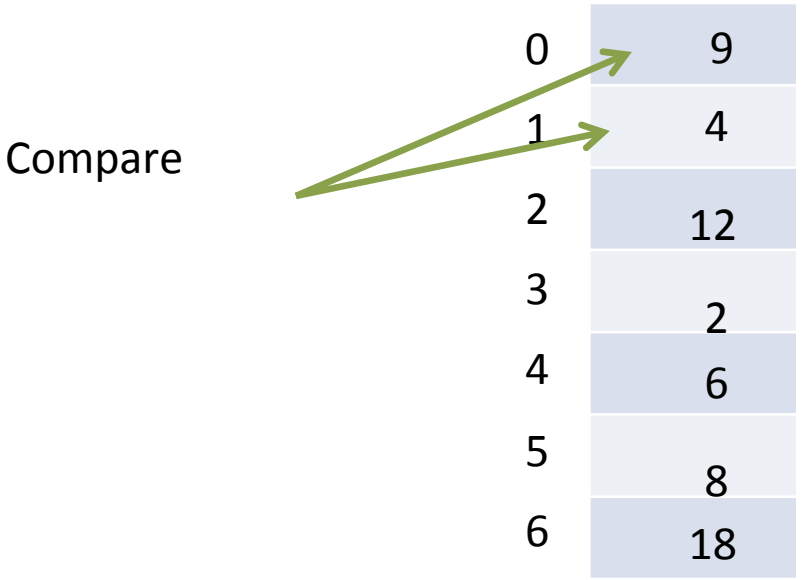


min = 0

```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
```


For all elements in the array

```
}
```

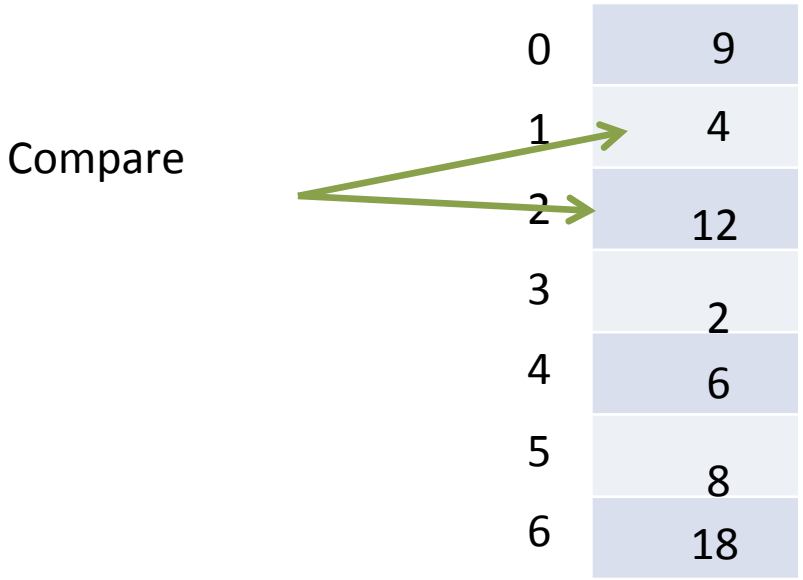


min = 0
j = 1
=> min = 1

```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
}
```

For all elements in the array

Find smallest in the remaining elements



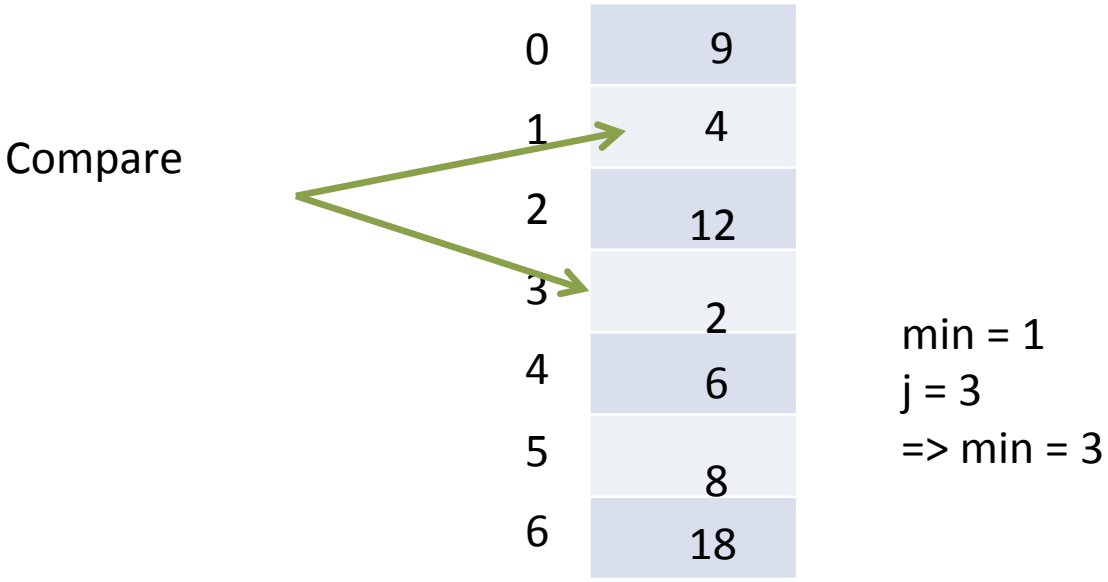

```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;



}
```

For all elements in the array

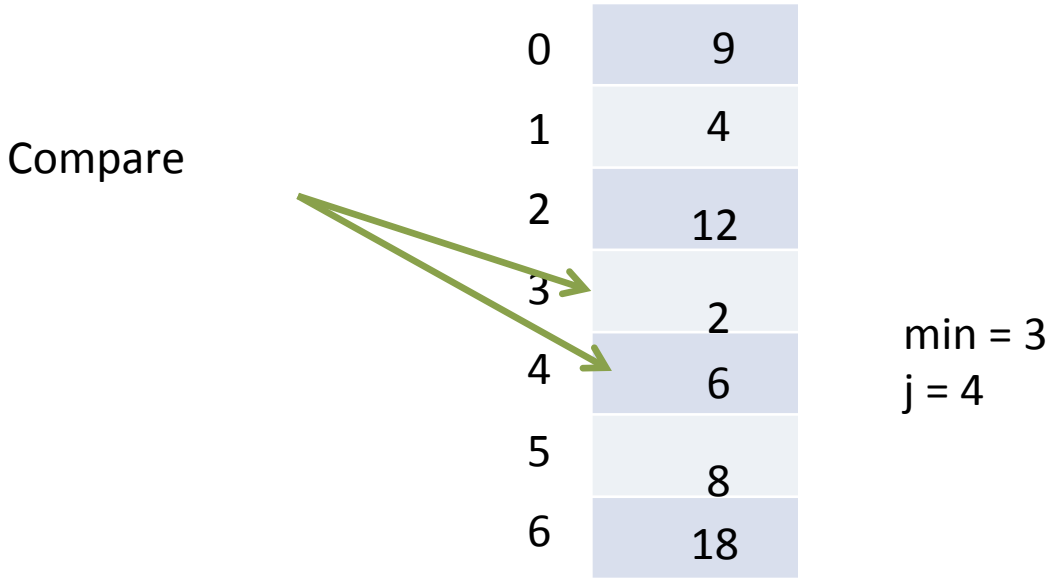
Find smallest in the remaining elements




```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
}
```

For all elements in the array

Find smallest in the remaining elements

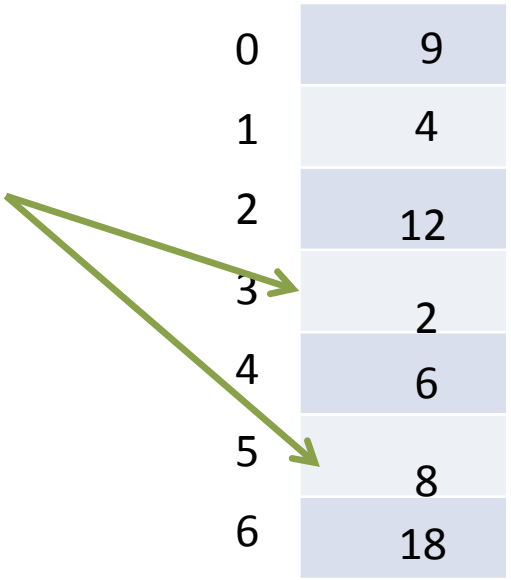



```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
}
```

For all elements in the array

Find smallest in the remaining elements

Compare



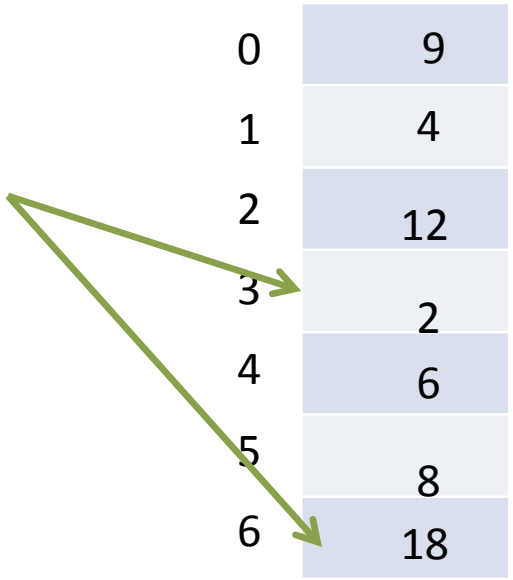
min = 3
j = 5

```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
}
```

For all elements in the array

Find smallest in the remaining elements

Compare



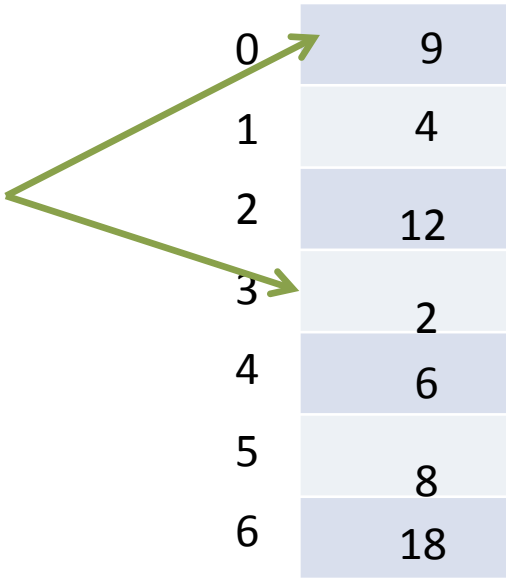
min = 3
j = 6

```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
}
```

For all elements in the array

Find smallest in the remaining elements

Swap



temp = 2

min = 3

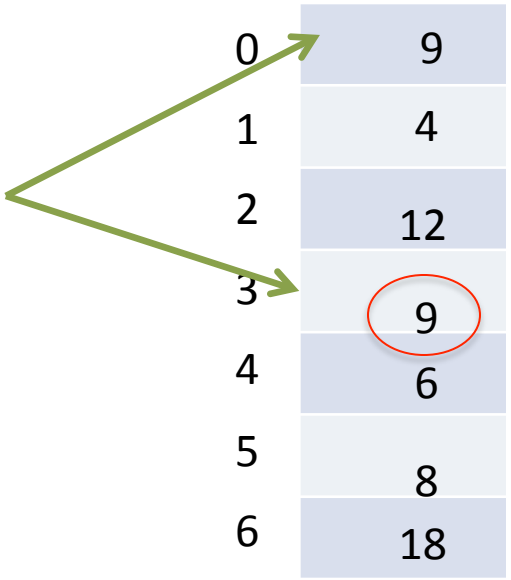
```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
        temp = numbers[min];
        numbers[min] = numbers[i];
        numbers[i] = temp;
}
```

For all elements in the array

Find smallest in the remaining elements

Swap

Swap



temp = 2

min = 3

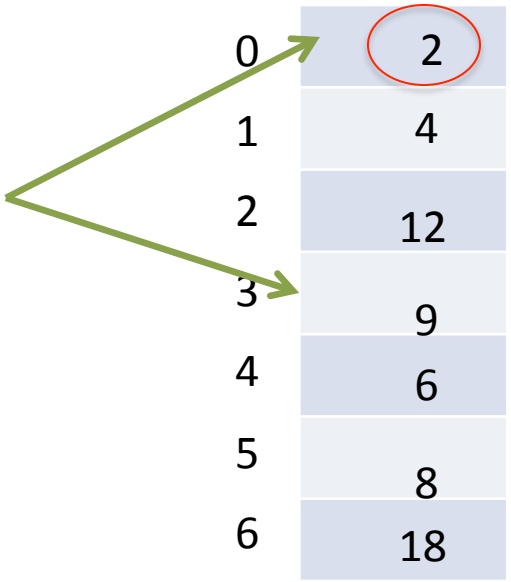
```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
        temp = numbers[min];
        numbers[min] = numbers[i];
        numbers[i] = temp;
}
```

For all elements in the array

Find smallest in the remaining elements

Swap

Swap



temp = 2

min = 3

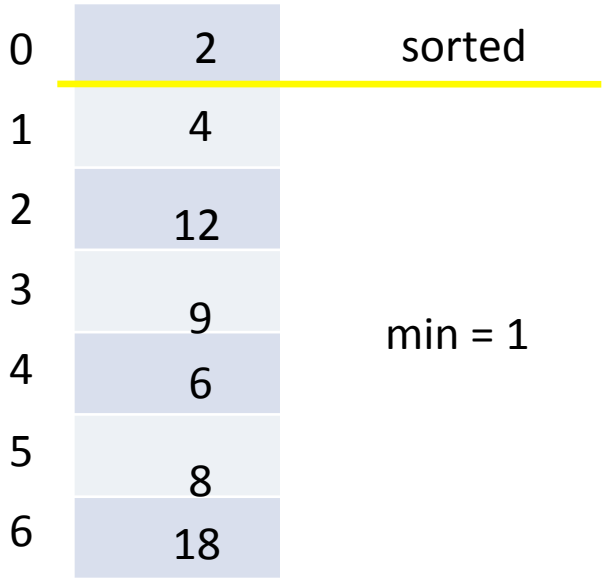
```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
        temp = numbers[min];
        numbers[min] = numbers[i];
        numbers[i] = temp;
}
```

For all elements in the array

Find smallest in the remaining elements

Swap

Repeat until outer loop is completed




```
for (int i = 0; i < numbers.length-1; i++) {
        min = i;
        for (int j = i+1; j < numbers.length; j++)
            if (numbers[j] < numbers[min])
                min = j;
        temp = numbers[min];
        numbers[min] = numbers[i];
        numbers[i] = temp;
}
```

For all elements in the array

Find smallest in the remaining elements

Swap

Polymorphism in Sorting

- Create a *generic* sorting algorithm
- Use the `Comparable` interface
  - Class implementing this interface has to define a `compareTo()` method to determine the relative order of its objects
- We can use **polymorphism** to develop a generic sort for any set of `Comparable` objects

How?

- The sorting method takes an array of `Comparable` objects as input parameter
- One method can be used to sort an array of `People`, or `Books`, or whatever
- This technique allows each class to decide for itself (via `compareTo()`) what it means for one object to be less than another

Generic Selection Sort

```
public static void selectionSort( Comparable[] list )
{
  int min;
  Comparable temp;

  // for each element in input list
  for( int i=0; i< list.length; i++ )
  {
    min = i;
    // find smallest element in the remainder of the list
    for( int j=i+1; j < list.length; j++ )
      if( list[j].compareTo( list[min] ) < 0 )
        min = j;

    // swap the values
    temp = list[min];
    list[min] = list[i];
    list[i] = temp;
  }
}
```

Generic Selection Sort

```
public static void selectionSort( Comparable[] list )
{
  int min;
  Comparable temp;

  // for each element in input list
  for( int i=0; i< list.length; i++ )
  {
    min = i;
    // find smallest element in the remainder of the list
    for( int j=i+1; j < list.length; j++ )
      if( list[j].compareTo( list[min] ) < 0 )
        min = j;

    // swap the values
    temp = list[min];
    list[min] = list[i];
    list[i] = temp;
  }
}
```

Generic Selection Sort

```
public static void selectionSort( Comparable[] list )
{
  int min;
  Comparable temp;

  // for each element in input list
  for( int i=0; i< list.length; i++ )
  {
    min = i;
    // find smallest element in the remainder of the list
    for( int j=i+1; j < list.length; j++ )
      if( list[j].compareTo( list[min] ) < 0 )
        min = j;

    // swap the values
    temp = list[min];
    list[min] = list[i];
    list[i] = temp;
  }
}
```

Steps to Writing Generic Sort Method

- Pass in generic input parameter
  - `void selectionSort( Comparable[] alist )`
- Use compareTo for sorting
  - `obj1.compareTo(obj2)`
- Any temp storage needs to be `Comparable`
  - `Comparable temp;`

Another Example

- Example: sorting an array of `Contact` objects
- See `PhoneList.java`
- See `Contact.java`

```
public class PhoneList
{
   //-----------------------------------------------------------------
   //  Creates an array of Contact objects, sorts them, then prints
   //  them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Contact[] friends = new Contact[8];

      friends[0] = new Contact ("John", "Smith", "610-555-7384");
      friends[1] = new Contact ("Sarah", "Barnes", "215-555-3827");
      friends[2] = new Contact ("Mark", "Riley", "733-555-2969");
      friends[3] = new Contact ("Laura", "Getz", "663-555-3984");
      friends[4] = new Contact ("Larry", "Smith", "464-555-3489");
      friends[5] = new Contact ("Frank", "Phelps", "322-555-2284");
      friends[6] = new Contact ("Mario", "Guzman", "804-555-9066");
      friends[7] = new Contact ("Marsha", "Grant", "243-555-2837");

      Sorting.selectionSort(friends);

      for (Contact oneFriend : friends)
         System.out.println (oneFriend);
   }
}
```

Static method

another for loop syntax

```
public class PhoneList
{
   //-----------------------------------------------------------------
   //  Creates an array of Contact objects, sorts them, then prints
   //  them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Contact[] friends = new Contact[8];

      friends[0] = new Contact ("John", "Smith", "610-555-7384");
      friends[1] = new Contact ("Sarah", "Barnes", "215-555-3827");
      friends[2] = new Contact ("Mark", "Riley", "733-555-2969");
      friends[3] = new Contact ("Laura", "Getz", "663-555-3984");
      friends[4] = new Contact ("Larry", "Smith", "464-555-3489");
      friends[5] = new Contact ("Frank", "Phelps", "322-555-2284");
      friends[6] = new Contact ("Mario", "Guzman", "804-555-9066");
      friends[7] = new Contact ("Marsha", "Grant", "243-555-2837");

      Sorting.selectionSort(friends);

      for (Contact oneFriend : friends)
         System.out.println (oneFriend);
   }
}
```

Output

```
Barnes, Sarah     215-555-3827
Getz, Laura       663-555-3984
Grant, Marsha     243-555-2837
Guzman, Mario     804-555-9066
Phelps, Frank     322-555-2284
Riley, Mark       733-555-2969
Smith, John       610-555-7384
Smith, Larry      464-555-3489
```

```
//********************************************************************
//  Contact.java       Author: Lewis/Loftus
//
//  Represents a phone contact.
//********************************************************************

public class Contact implements Comparable
{
   private String firstName, lastName, phone;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this contact with the specified data.
   //-----------------------------------------------------------------
   public Contact (String first, String last, String telephone)
   {
      firstName = first;
      lastName = last;
      phone = telephone;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  First name accessor.
   //-----------------------------------------------------------------
   public String getFirstName (){return firstName;}

   //-----------------------------------------------------------------
   //  Last name accessor.
   //-----------------------------------------------------------------
   public String getLastName () {return lastName;}

   //-----------------------------------------------------------------
   //  Returns a description of this contact as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      return lastName + ", " + firstName + "\t" + phone;
   }

   //-----------------------------------------------------------------
   //  Returns a description of this contact as a string.
   //-----------------------------------------------------------------
   public boolean equals (Object other)
   {
      return (lastName.equals(((Contact)other).getLastName()) &&
              firstName.equals(((Contact)other).getFirstName()));
   }
```

continue

continue

```
    //-----------------------------------------------------------------
    //  Uses both last and first names to determine ordering.
    //-----------------------------------------------------------------
    public int compareTo (Object other)
    {
        int result;

        String otherFirst = ((Contact)other).getFirstName();
        String otherLast = ((Contact)other).getLastName();

        if (lastName.equals(otherLast))
            result = firstName.compareTo(otherFirst);
        else
            result = lastName.compareTo(otherLast);

        return result;
    }
}
```

```
public class Demo
{
   //-----------------------------------------------------------------
   //  Consider second call to selectionSort()
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Contact[] friends = new Contact[4];

      friends[0] = new Contact ("John", "Smith", "610-555-7384");
      friends[1] = new Contact ("Sarah", "Barnes", "215-555-3827");
      friends[2] = new Contact ("Mark", "Riley", "733-555-2969");
      friends[3] = new Contact ("Laura", "Getz", "663-555-3984");

      Sorting.selectionSort(friends);
      for (Contact oneFriend : friends)
         System.out.println (oneFriend);

      int[] numbers = {3, 9, 4, 7, 2, 6};
      Sorting.selectionSort(numbers);
   }
}
```

Same as before

Will this work with the generic definition?

Using Generic Sort Method

- Define your own class that does the following
- Implement `Comparable`
  - E.g.:

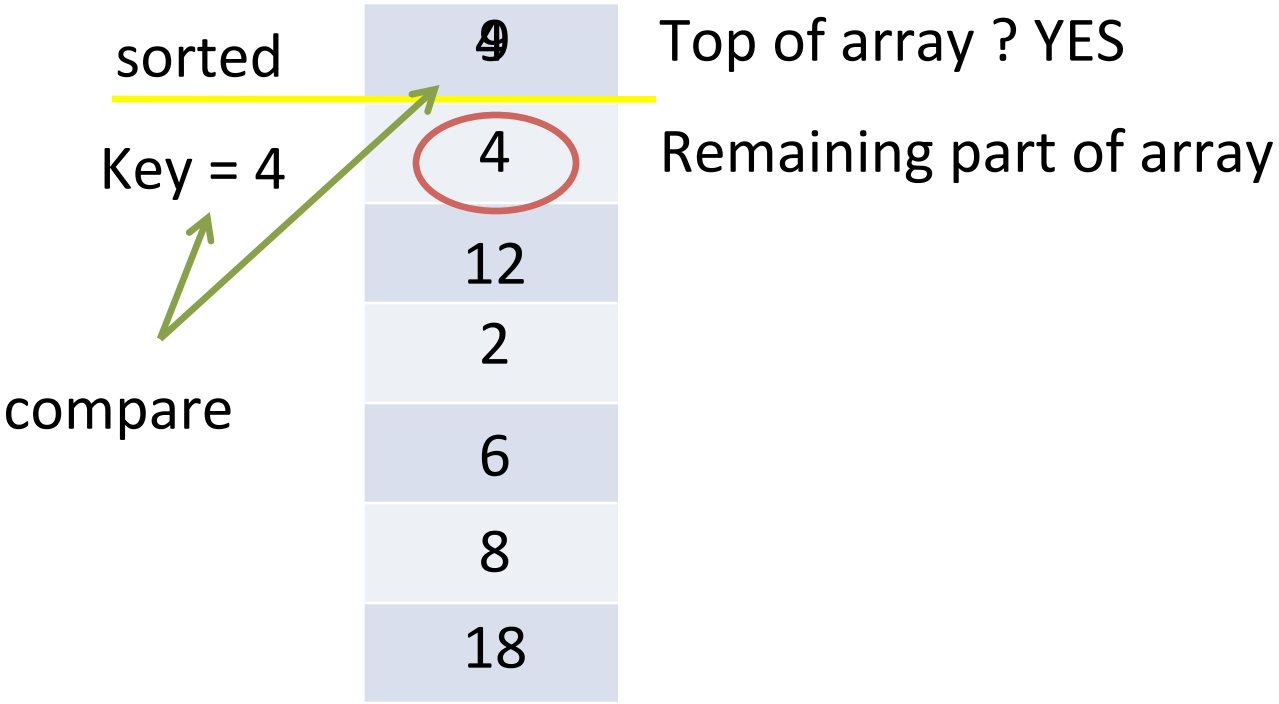
```
public class Contact implements Comparable
```

- Override `compareTo` method
  - `public int compareTo( Object other )`
- Override `equals` method
  - `public boolean equals( Object other )`

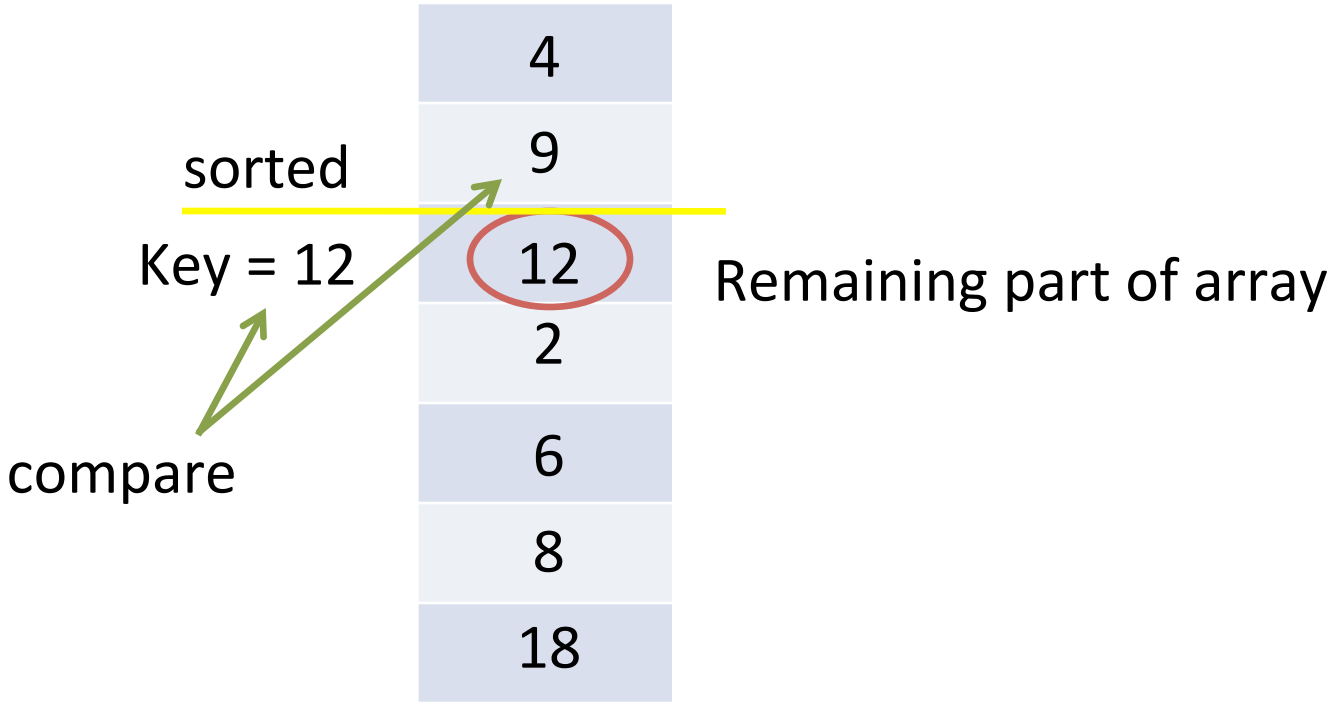
Insertion Sort

- The strategy of Insertion Sort:
  - Build a sorted sublist of one item: take the first item
  - Insert the second item into the sorted sublist, shifting the first item as needed to make room to insert the new one
  - Insert the third item into the sorted sublist (of two items), shifting items as necessary
  - Repeat until all values are inserted into their proper positions

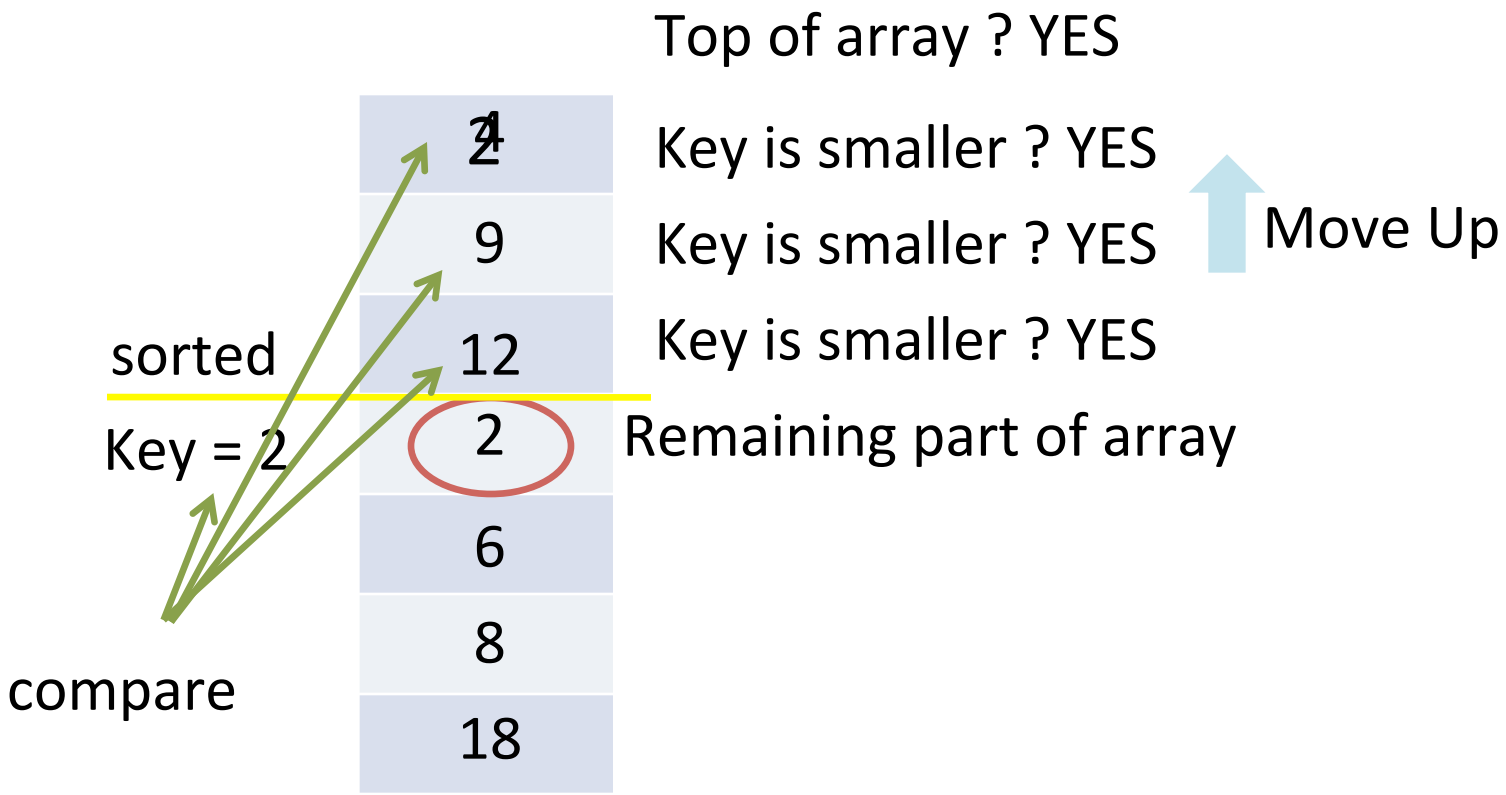
Insertion sort – Round 1



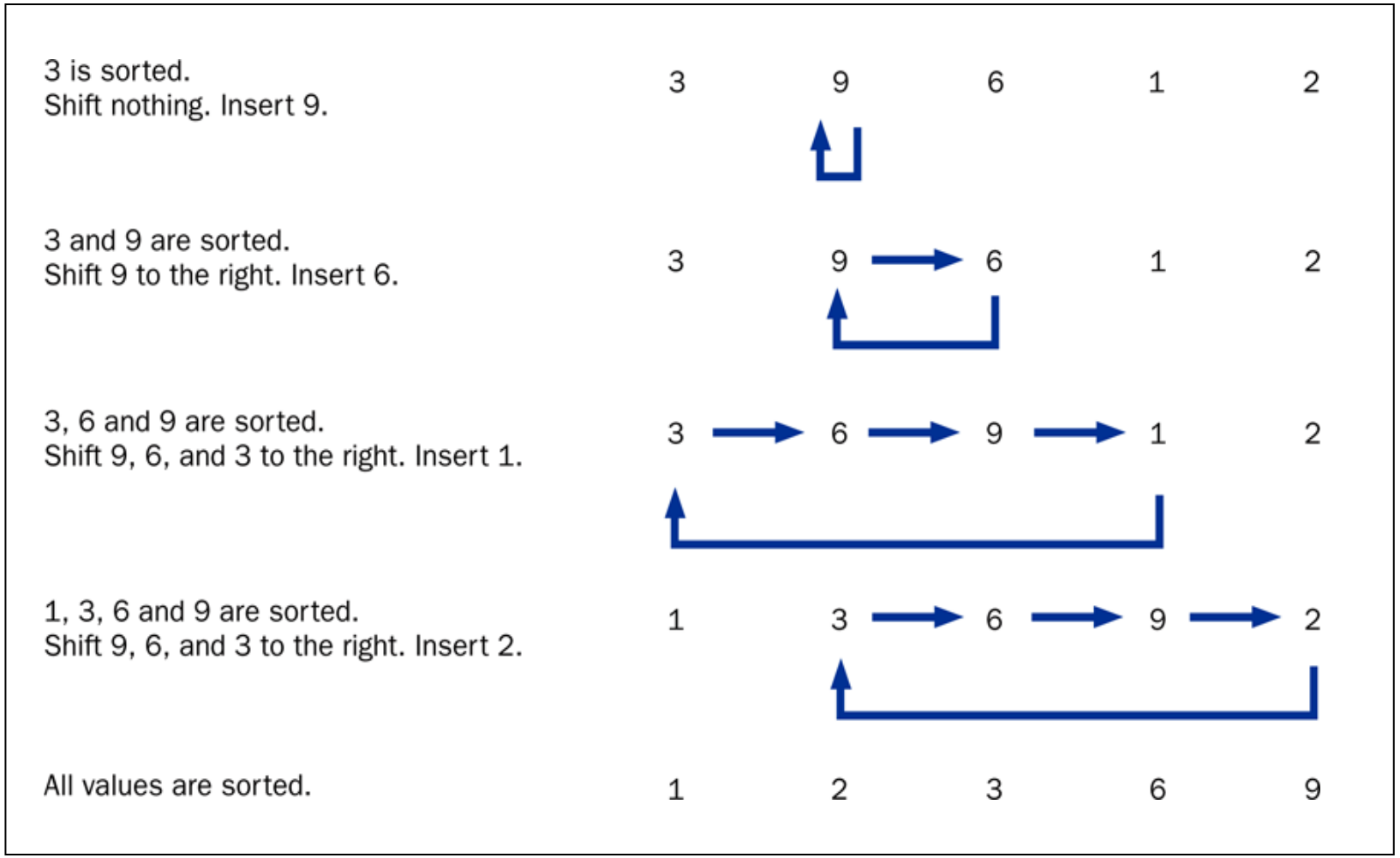
Insertion sort – Round 2



Insertion sort – Round 3



Insertion Sort



Defining Insertion Sort

```
public static void insertionSort( int[] list )
{
  //  1. created sorted sublist: item one is sorted
  //  2. for each remaining element in the list

  //  a. compare element with those in sorted sublist

  //  b. keep shifting larger values to the right

  //  c. insert element when no larger value or
  //     when top of array is reached

}
```

Summary

- Introduction to sorting algorithms
  - Major difficulties
  - Measure of efficiency
- Selection sort intuition and implementation
- Implementing generic sort methods
  - `Comparable` interface
  - `Object` class and casting
  - Polymorphism
- Insertion sort intuition
- Next: insertion sort code + searching algorithms