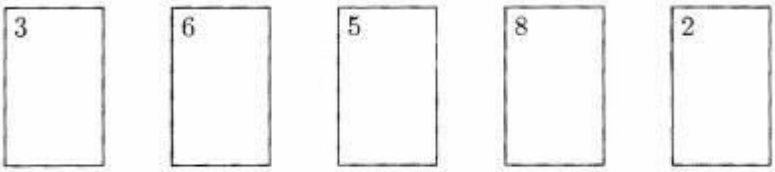
10.3 Insertion Sort

To use a binary search, we noted that it was necessary to have the items sorted. We now turn our attention to finding ways of sorting data. To develop algorithms of any kind, it is often useful to examine the ways that we solve similar problems in everyday life. To develop a solution to the problem of sorting data, let us look at how we might proceed if we were playing a card game and we wanted to put our hand in order, from smallest to largest. A common way of sorting such a hand is to examine cards from left to right, rearranging them if necessary by placing cards where they belong in the sequence of cards on their left. The next example illustrates the process.

Example 1

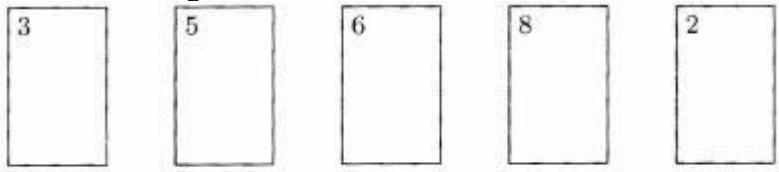
Suppose that we have the five cards shown below (ignore suits).

We could begin to sort these cards by looking at the 3, the second card from the left. Since 3 < 6, the 3 should be inserted to the left of the 6. This will produce the following arrangement in which the two values on the left are in order.



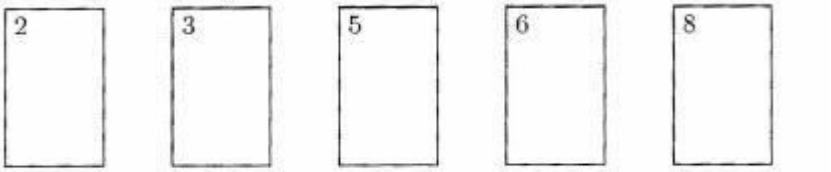
The next card from the left, the 5, should be inserted between the 3 and the 6. Doing this gives us the next arrangement in which the first three cards are guaranteed to be in order.

Now we examine the 8. Because it is greater than any of the values to its left, it should stay where it is and still give us the four leftmost values in order.



Finally, looking at the 2, we can see that it must be inserted before any of the other values.

Doing this gives us our final, ordered arrangement.



To implement this algorithm for an array of values, we must examine and correctly insert each of the values in the array from the second position up to the last one. This requires a loop like the following, for an array called `list`.

```
for (int top = 1; top < list. length; top++)
 // insert element at top into its correct position
 // among the elements from 0 to top - 1
```

At each stage or pass of the sort, we first copy the element that we want to insert into a temporary location. We then move from right to left through the (already sorted) items on the left of the value that we are inserting. If a value in the list is larger than the new one, it is moved one space to the right (to provide room for the new item). Once the correct location of the new value is found, it is then inserted back into the array from the temporary location in which it had been saved. These ideas are incorporated into the method of the next example.

Example 2

The method `insertSort` uses an insertion sort to arrange an array of double values in ascending order.

```
public static void insertSort (double[] list)
{
  for (int top = 1; top < list.length; top++)
  {
    double item = list [top];
    int i = top;
    while (i > 0 && item < list[i-1])
    {
      list[i] = list [i-1] ;
      i--;
    }
    list[i] = item;
  }
}
```

Exercise 10.3

1. An insertion sort is to be used to put the values

   6 2 8 3 1 7 4

in ascending order. Show the values as they would appear after each pass of the sort.

Pass 1	2	6	8	3	1	7	4
Pass 2	2	6	8	3	1	7	4
Pass 3	2	3	6	8	1	7	4
Pass 4	1	2	3	6	8	7	4
Pass 5	1	2	3	6	7	8	4
Pass 6	1	2	3	4	6	7	8

```
//TOP = 1
//6,6,8,3,1,7,4
//2,6,8,3,1,7,4

//TOP = 2
//2,6,8,3,1,7,4

//TOP = 3
//2,6,8,8,1,7,4
//2,6,6,8,1,7,4
//2,3,6,8,1,7,4

//TOP = 4
//2,3,6,8,8,7,4
//2,3,6,6,8,7,4
//2,3,3,6,8,7,4
//2,2,3,6,8,7,4
//1,2,3,6,8,7,4

//TOP = 5
//1,2,3,6,8,8,4
//1,2,3,6,7,8,4

//TOP = 6
//1,2,3,6,7,8,8
//1,2,3,6,7,7,8
//1,2,3,6,6,7,8

//FINAL
//1,2,3,4,6,7,8
```

```
public class Question1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {6,2,8,3,1,7,4};
		insertSort(list);
	}

	public static void insertSort (double[] list)
	{
```

```
        for (int top = 1; top < list.length; top++)
        {
          double item = list [top];
          int i = top;
          while (i > 0 && item  < list[i-1])
          {
            list[i] = list [i-1];
            i--;
          }
          list[i] = item;
        }
      }
}
```

2. What changes would have to be made to the `insertSort` method in Example 2 in order to sort the values in descending order?

   You would have to change the sing of `item  < list[i-1]` to `item  > list[i-1]`

```
public class Question2 {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            double [] list = {6,2,8,3,1,7,4};
            insertSort(list);
            for (int x = 0; x < list.length; x++)
                  System.out.print(list[x] + "\t");
      }

      public static void insertSort (double[] list)
      {
        for (int top = 1; top < list.length; top++)
        {
          double item = list [top];
          int i = top;
          while (i > 0 && item  > list[i-1])
          {
            list[i] = list [i-1] ;
            i--;
          }
          list[i] = item;
        }
      }
}
```

3. What might happen if, in Example 2, the `while` statement's first line were written in the following form?

```
while (item < list[i-1] && i > 0)
```

Since java checks its conditions going from left to right, for case i = -1, this statement would not evaluate the second statement because the first statement would throw a `java.lang.ArrayIndexOutOfBoundsException` exception.

4. Write a program that initializes an array with the names of the planets ordered by their distances from the sun (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto) and prints them in that order on one line. The program should then use an insertion sort to arrange the names alphabetically. To trace the progress of the sort, have it print the list after each pass.

```
public class Question4 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		String [] planets = {"Mercury", "Venus", "Earth", "Mars",
"Jupiter", "Saturn", "Uranus", "Neptune","Pluto"};
		insertSort (planets);
		System.out.println("ORDERED LIST OF PLANETS:");
		for (int x = 0; x < planets.length; x++)
			System.out.println(planets[x]);


	}
    public static void insertSort (String [] list) {

	for (int x = 1; x < list.length; x++) {
			String item = list[x];
			int i = x;

			while (i > 0 && item.charAt(0) < list[i-1].charAt(0))
{
				list [i] = list[i-1];
				i--;
			}
			list[i] = item;
			System.out.println("PASS " + x);
			for (int y = 0; y < list.length; y++)
				System.out.print(list[y] + " ");
			System.out.println();
		}
    }

}
```

5. The *median* of an ordered list of numerical values is defined in the following way. If the number of values is odd, the median is the middle value. If the number of values is even, the median is the average of the two middle values. Write a program that first prompts the user for the number of items to be processed, reads that many real values, and then finds their median.

```
import java.util.Scanner;
public class Question5 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner input = new Scanner (System.in);
		System.out.println("Enter the number of items to be
processed:");
		int items = input.nextInt();
		double [] list = new double[items];
		for (int x = 0; x < list.length; x++) {
			System.out.println("Enter value " + (x+1));
			list[x] = input.nextInt();
		}

		insertSort(list);
		System.out.println("The median of this list is " +
Median(list));

	}

	public static void insertSort (double [] list) {
		for (int x = 0; x < list.length; x++) {
			double item = list[x];
			int i = x;
			while (i > 0 && item < list[i-1]) {
				list[i] = list[i-1];
				i--;
			}
			list[i] = item;
		}
	}

	public static double Median (double [] list) {
		double median;
		if (list.length < 1 )
			median = 0;
		else if (list.length % 2 == 0)
			median = (list[list.length/2] + list[(list.length/2) -
1])/2;
		else
```

```
                median = list[(list.length-1)/2];
            return median;
        }

}
```

6. A sort is said to be *stable* if it always leaves values that are considered to be equal in the same order after the sort. Is the insertion sort stable? Justify your answer.

   The insertion is stable because it leaves the values in the same order after the sort. This is because the `while` loop statement only runs if `item < list[i-1]` hence if `item` is equal to `list[i-1]`, their order after the sort will not switch whatsoever.

```
public class Question6 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                double [] list = {2,3,4,2,4,3,10,1};
                insertSort(list);
                for (int x = 0; x < list.length; x++)
                        System.out.print(list[x] + " \t");


        }

        public static void insertSort (double [] list) {
                for (int x = 0; x < list.length; x++) {
                        double item = list[x];
                        int i = x;
                        while (i > 0 && item < list[i-1]) {
                                list[i] = list[i-1];
                                i--;
                        }
                        list[i] = item;
                }
        }

}
```