Searching & Sorting in Java – Insertion Sort

Insertion Sort

The insertion sort algorithm focuses on a single element of the list and tries to find the correct location for that element. When searching for the correct position, only positions below the current position are considered. Thus, on the first pass, the first element is always in the correct position (so it is redundant to perform this step).

As is often the case, this is best illustrated through an example.

6	3	5	8	2

Pass #1:

6	3	5	8	2

- compare 3 to all values below it (to the left); if they are larger, shift them right
- 6 is larger than 3, so move 6 to position 1
- put 3 in the empty location (which is 0).

3	6	5	8	2

Pass #2:

3	6	5	8	2

- compare all values to 5
- 5 is less than 6, so shift 6 right to position 2
- 5 is greater than 3, so stop; put 5 in position 1

3	5	6	8	2

Pass #3:

- no change; 8 is already in the correct location

3	5	6	8	2

Pass #4:

3	5	6	8	2

- compare all values to 2
- 2 is less than each value below it, so they will be shifted right by one after each comparison
- when the bottom of the list is reached, the pass ends, and 2 is put at the available location at the beginning

2	3	5	6	8

Searching & Sorting in Java – Insertion Sort

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.

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

For each pass, we need to copy the element under consideration into a temporary location. Now, as we move from right to left through the array, we shift any larger values to the right. Once we have moved any larger values, we put our target element into the last empty location.

```
public static void insertSort( double[] list)
{
      for (int top = 1; top < list.length; top++)
      {
            double item = list[top];   // temporary storage of item
            int i = top;

            while (i > 0 && item < list[i-1])
            {
                  // shift larger items to the right by one
                  list[i] = list[i-1];
                  // prepare to check the next item to the left
                  i--;
            }
            list[i] = item;       // put sorted item in open location
      }
}
```

Exercises

1. An insertion sort is to b used to put the values 6 2 8 3 1 7 4 in ascending order (lowest to highest). Show the values as they would appear after each pass of the sort.

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

3. What might happen if the while statement's first line were written in the following form?
   while (item < list[i-1] && i > 0)

4. Write a program that initializes an array with the names of the planets in their typical order (from closest to furthest from the Sun) and prints them in that order on one line. The program should then use an insertion sort algorithm to arrange the names alphabetically. To trace the progress of the sort, have it print the list after each pass.

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 prompts the user for the number of items to be processed, reads that many real values, and then finds their 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.