17.02 Virtual Lecture Notes

Bubble sort is not the only way to sort an array. In fact, there are many ways to sort an array. Another sorting algorithm is the **insertion sort**. To perform insertion sort, you create an empty copy of the array you are sorting. Then, one by one, you take values from the array you want to sort and insert them into the empty array. Insertion gets it name from the fact that, as you are inserting elements into the empty array, you look to see where the next element should go and move array elements to the right as necessary.

Let us take a look at the algorithm as it would work on an array of integers:

```
int[] source = { 12, 14, 15, 11, 13 };
int[] dest = new int[ source.length ];

for ( int i = 0 ; i < source.length ; i++ )
{
  int next = source[ i ];
  int insertindex = 0;
  int k = i;
  while ( k > 0 && insertindex == 0 )
  {
    if ( next > dest[ k - 1 ] )
    {
      insertindex = k;
    }
    else
    {
      dest[ k ] = dest[ k - 1 ];
    }
    k--;
  }

  dest[ insertindex ] = next;
}
```

The **dest** array is an empty array. The **for** loop steps through the source array and inserts each item, one at a time, into the destination array. It accomplishes this by taking the item to be inserted (**next**) and finding where to insert it by going through the destination array (**dest**) to find the proper spot. Notice that **k** is set to **i** so that we do not look at entries in **dest** that have no values. Working backwards in **dest**, we keep comparing next to a location in **dest**. If it is **>**, then we have found the spot to add it. Otherwise, we copy the element at location **k-1** to **k** (move it right), so that we are prepared for inserting the **next** into its spot. If we did not move **k-1**, then we would have to move it after we found the location for **next**. By moving it as we find the location, we save time in the long run.

Now, let us put this into a method. We have two choices: either we pass two arrays to the method (the one to be sorted and an empty one), or we pass one array and have the method return a sorted array.

If we chose the first method, we would have this:

```
   public static void insertionSort(HouseListing[] source,
                                    HouseListing[] dest)
   {

       for ( int i = 0 ; i < source.length ; i++ )
       {
           HouseListing next = source[ i ];
           int insertindex = 0;
           int k = i;
           while ( k > 0 && insertindex == 0 )
           {
              if ( next.getCost() > dest[k-1].getCost() )
              {
                  insertindex = k;
              }
              else
              {
                 dest[ k ] = dest[ k - 1 ];
              }
               k--;
           }

           dest[ insertindex ] = next;
       } // end of for
   }
```

Notice that this will sort our list of houses based on cost and in ascending order. Again, to get descending order, just modify the comparison of the **`if`** statement to use a <.

If we choose the second way, our method will look like this:

```
   public static HouseListing[] insertionSort(HouseListing[]
                                                        source)
   {
       HouseListing[] dest = new HouseListing[ source.length ];

       for ( int i = 0 ; i < source.length ; i++ )
       {
           HouseListing next = source[ i ];
           int insertindex = 0;
           int k = i;
```

```
            while ( k > 0 && insertindex == 0 )
            {
               if ( next.getCost() > dest[k-1].getCost() )
               {
                   insertindex = k;
               }
               else
               {
                  dest[ k ] = dest[ k - 1 ];
               }
                k--;
            }

            dest[ insertindex ] = next;
         } // end of for
         return dest;
   }
```

Notice that this time we pass **`dest`** back via the return statement.

Both methods perform the sort we desire, but the first would require you to have two arrays in the main program. Therefore, many people would choose the second way. However, what if you wanted to keep the original array in its original order for later use? The first method would allow us to do that. So depending on what you want to accomplish with the original array, you could choose either method. In the demo program **`TestListing2.java`**, we use the second method. **`TestListing3.java`** uses the first.

- Download the TestListing2.java and TestListing3.java files to your module 17 demo programs directory and open them.
- Run the files and make sure you understand them before you continue.