Bubble Sort Code

```java
public static void bubbleSort(Comparable[] theArray, int n) {

  // ---------------------------------------------------
  // Sorts the items in an array into ascending order.
  // Precondition: theArray is an array of n items.
  // Postcondition: theArray is sorted into ascending
  // order.
  // ---------------------------------------------------
    boolean sorted = false; // false when swaps occur
    for (int pass = 1; (pass < n) && !sorted; ++pass) {
    // Invariant: theArray[n+1-pass..n-1] is sorted
    // and > theArray[0..n-pass]
      sorted = true; // assume sorted
      for (int index = 0; index < n-pass; ++index) {
      // Invariant: theArray[0..index-1] <= theArray[index]
        int nextIndex = index + 1;
        if (theArray[index].compareTo(theArray[nextIndex]) > 0) {
        // exchange items
          Comparable temp = theArray[index];
          theArray[index] = theArray[nextIndex];
          theArray[nextIndex] = temp;
          sorted = false; // signal exchange
        } // end if
      } // end for
    // Assertion: theArray[0..n-pass-1] < theArray[n-pass]
    } // end for
} // end bubbleSort
```

At most (n-1) passes

exchanges

Pass1 requires (n-1) comparisons and at most (n-1) exchanges

Pass2 requires (n-1) comparisons and at most (n-2) exchanges

⋮

In general pass i requires (n-i) comparisons and at most (n-i) exchanges.

The worst case:

$(n-1) + (n-2) + \cdots + 1 = n*(n-1)/2$ comparisons

and $(n-1) + (n-2) + \cdots + 1 = n*(n-1)/2$ exchanges

Each exchange requires 3 data moves.

$$\text{Total} = \underbrace{3*(n)*(n-1)/2}_{\text{exchanges}} + \underbrace{n*(n-1)/2}_{\text{comparisons}}$$

$$\frac{4(n)(n-1)}{2} = 2n^2 - 2 \quad \boxed{O(n^2)} \text{ Worst case}$$

Best case occurs when the original array is already sort. One pass (n-1) comprisons. and no exchanges $\boxed{O(n)}$