SORTING: Bubble sort

- The serial bubble sort algorithm for sorting a list of integers can be implemented as follows:

```
for (list_length = n; list_length>= 2; list_length--)
for (i = 0; i < list_length-1; i++)
if (a[i] > a[i+1])
{
        tmp = a[i];
        a[i] = a[i+1];
        a[i+1] = tmp;
}
```

- Here, a stores ***n***ints and the algorithm sorts them in increasing order. The outer loop first finds the largest element in the list and stores it in a[n-1]; it then finds the next-to-the-largest element and stores it in a[n-2], and so on. So, effectively, the first pass is working with the full n-element list. The second is working with all of the elements, except the largest; it's working with an n-1-element list, and so on.

- The inner loop compares consecutive pairs of elements in the current list. When a pair is out of order (a[i] > a[i+1]) it swaps them. This process of swapping will move the largest element to the last slot in the "current" list, that is, the list consistingof the elements a[0], a[1], . . . , a[list_length-1]

- There's a loop-carried dependence in the outer loop; in any iteration of the outer loop the contents of the current list depends on the previous iterations of the outer loop. For example, if at the start of the algorithm a = 3, 4, 1, 2, then the second iteration of the outer loop should work with the list 3, 1, 2, since the 4 should be moved to the last position by the first iteration. But if the first two iterations are executing simultaneously, it's possible that the effective list for the second iteration will contain 4.
- The loop-carried dependence in the inner loop is also fairly easy to see. In iteration i the elements that are compared depend on the outcome of iteration i- 1. If in iteration i-1, a[i-1] and a[i] are not swapped, then iteration i should compare a[i] and a[i+1]. If, on the other hand, iteration i-1 swaps a[i-1] and a[i], then iteration i should be comparing the original a[i-1] (which is now a[i]) and a[i+1].
- It's important to keep in mind that even though we can always find loop-carried dependences, it may be difficult or impossible to remove them. The parallel for directive is not a universal solution to the problem of parallelizing for loops.