[Overall Information]

This is the first class note for the first class. Welcome to CSCI303.
This class gave a course outline, course requirements, and grading policy of CSCI303 at Spring 2006.
It then discussed the question of what algorithms are, and how they fit in with the rest of computer science. Following, we saw a first example of how to prove an algorithm correct, in this case Insertion Sort.

[During the Lecture]

1. Announcements

1. To learn more about current topics in computer science, students are encouraged to attend the Computer Science Colloquium.

2. Regular office hours Will be every Tuesday from 10:30 to 12:00 at SAL232. Both of the two TAs will have regular office hours too. The time and places will be announced shortly.

3. The final exam, which will probably on May 08 from 11:00-13:00
(please check your schedule for possible conflict), will count for 40% of the final grades, the midterm is for 30% and quizzes for the last 30%.

4. Submission of homework is optional (if the homework is submitted, the TAs will provide feedback). Homework is where one learns material the best.

5. Class participation will not be graded, but is strongly encouraged.

2. First question for this class: what is Computer Science? What do we learn as a Computer Science major? The answer is: solving problems with and about computers. The definition of an algorithm is also a way to solve a problem. The difference between what is taught in a course titled
“Algorithms” and other courses is that the algorithms covered in CSCI303 are applicable to many different areas and problems within computer science, and are less specific to one particular sub-area.

The next question is: what is the difference between programming and algorithm? Programming is the detailed implementation of algorithm. For designing and analyzing algorithms we are able to omit programming details, in order to focus on the important ideas.

Both of these suggest that algorithms in this class will be discussed in more abstract terms: frequently, we will be reasoning about graphs, arrays, strings, formulas, etc.

Thus, an important notion (which will be central throughout the course) is that of “reduction”: showing how to solve one problem using the solution of another. In particular, many real-world problems can be reduced to the abstract ones we study.

A list of the topics which will (hopefully) be covered is given:
4. After the introduction part, the class moved to Sorting Problem. To introduce this problem, a specific example of sorting number array was written onto the whiteboard.

The first algorithm solving the Sorting Problem we were going to study is Insertion Sort. We first saw a specific example to illustrate the idea of Insertion Sort and wrote the abstracted algorithm down afterwards.
(Because we study algorithms in general, for CSCI303’s classes, quizzes, midterm and final exams, you can write down your algorithm in any format you like as long as they are clear enough to understand.)

The discussion then led to the observation that the most important two properties of algorithms are correctness and speed. Additional considerations include simplicity and brevity, as well as memory consumption.

5. When we have come up with an algorithm, it is important to show that it actually works. Testing can discover errors, but cannot guarantee that
there are no errors. To make sure that an algorithm is correct, we can use mathematical proof techniques. Indeed, correctness proofs of algorithms will be one of the main things we learn in CS303.

Our goal at this moment is to prove the correctness of Insertion Sort. The key statement here is that after i iterations, the first i elements in the array are sorted. However, it should also be noted crucially that we need to state/ensure that the elements in the array are still the same that they were, i.e., that nothing was overwritten. The formal proof for this statement using induction was written onto the whiteboard in details in the lecture.

6. The key step of the proof is: assuming the first i elements are sorted after the first i iterations. During the (i+1) iteration, the (i+1) element will be swapped into exactly the right position without affecting the
already sorted parts of the array. To prove that the element is swapped into exactly the right position, we use the fact that the inner while loop terminated. Therefore, the element must be at least as large as the one to its left, or it must be at the left end of the array. Also, an element gets only swapped to the left if it is smaller than its left neighbor. Therefore, it is always smaller than its right neighbor (after the swapping), or at the rightmost end (if it didn't get swapped at all).

In addition, there is no overwriting happened because the only operation performed is swap on the left elements, and the right elements (right of position i+1) are never touched.

Taking all this together, because the element i+1 was inserted in the right place, and the rest was already sorted by induction hypothesis, the first i+1 elements are sorted after the iteration i+1.