

# Class Note #02

Date: 01/11/2006

## [Overall Information]

In this class, after a few additional announcements, we study the worst-case running time of Insertion Sort.

The asymptotic notation (also called, Big-O notation) which was introduced in today's class is a very important concept we will frequently use throughout this semester.

## [During the Lecture]

1, After the class began, a few announcements were made:

(1) Students clearly need to do the homework but do not need to hand the homework in if they do not want feedback. Homework will not contribute to the final grades of CSCI303@Fall2005. However, please be aware that some of the quiz questions maybe similar or identical to homework questions while the quiz grading does contribute to the final grades.

(2) Midterm and final exam will be open book and open notes. You are free to bring your own class notes at the exams. The grading will not be curved.

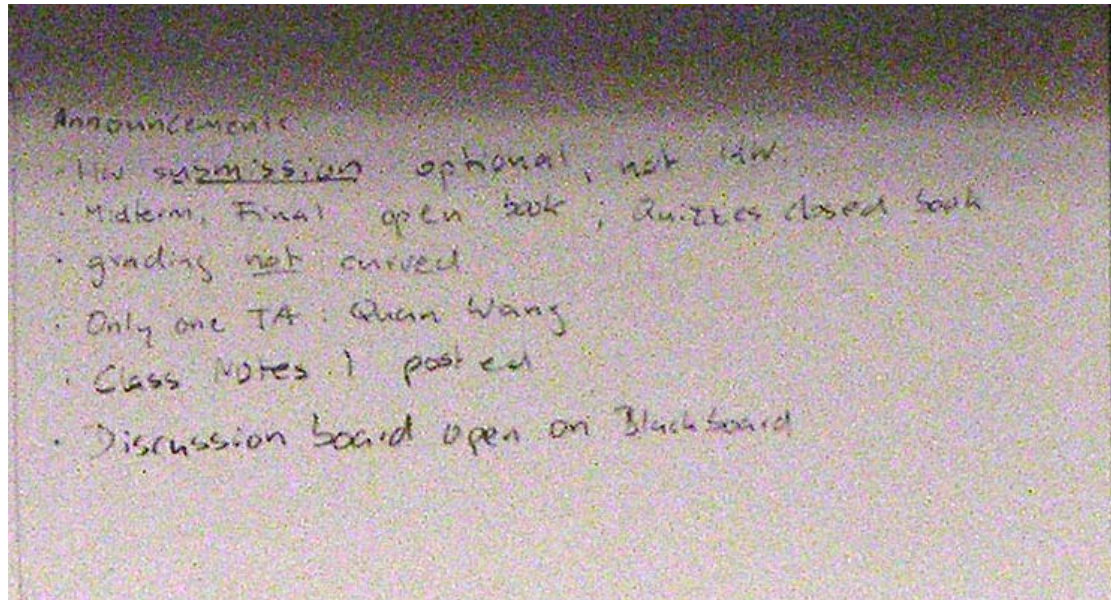
(3) Typically, class notes will be posted one or two days later after each lecture at the following address:

<http://www-rcf.usc.edu/~dkempe/CS303/index.html#notes>

All the class notes are being recorded and written along with the actual lectures. So what I can guarantee is the content is up to date while what I

am not able to guarantee is those notes' completeness and absolutely correctness. Please look those class notes as another reference source.

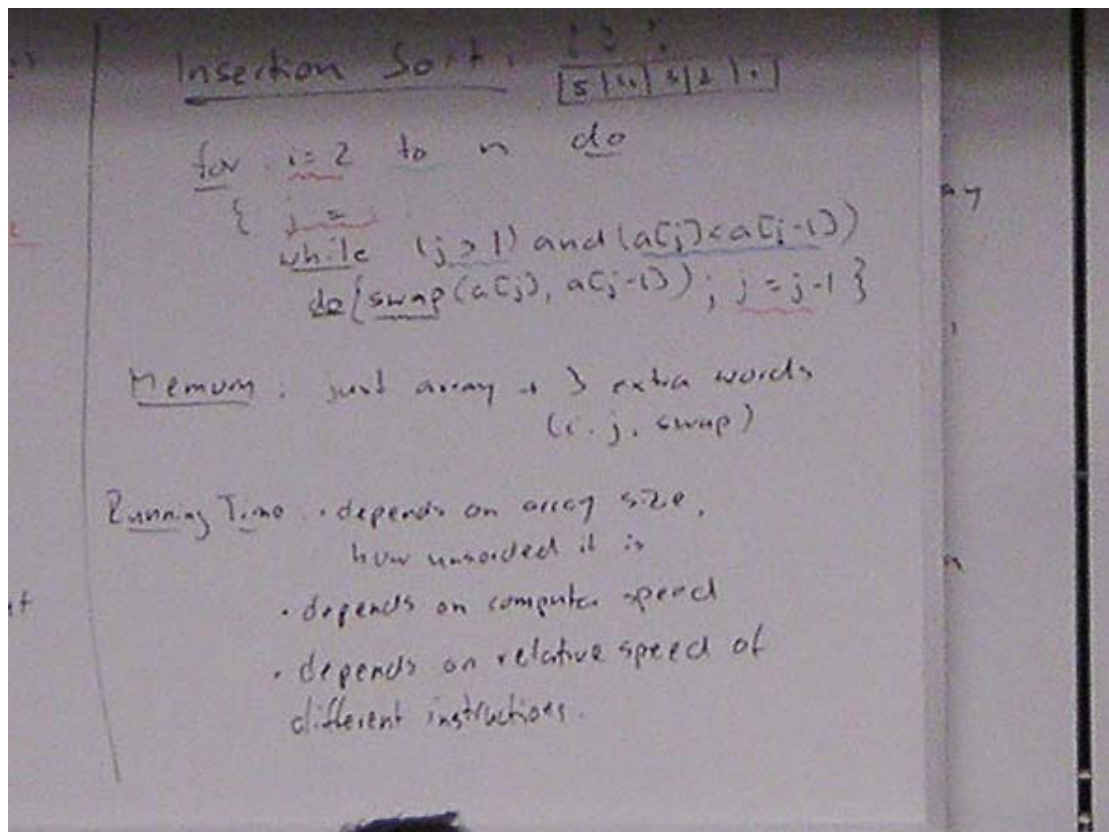
(4) Using the discussion board (located at "<https://totale.usc.edu/>") to discuss course related questions is highly encouraged.



2. After those announcements, we reviewed what we learnt about Insertion Sort in the last lecture.

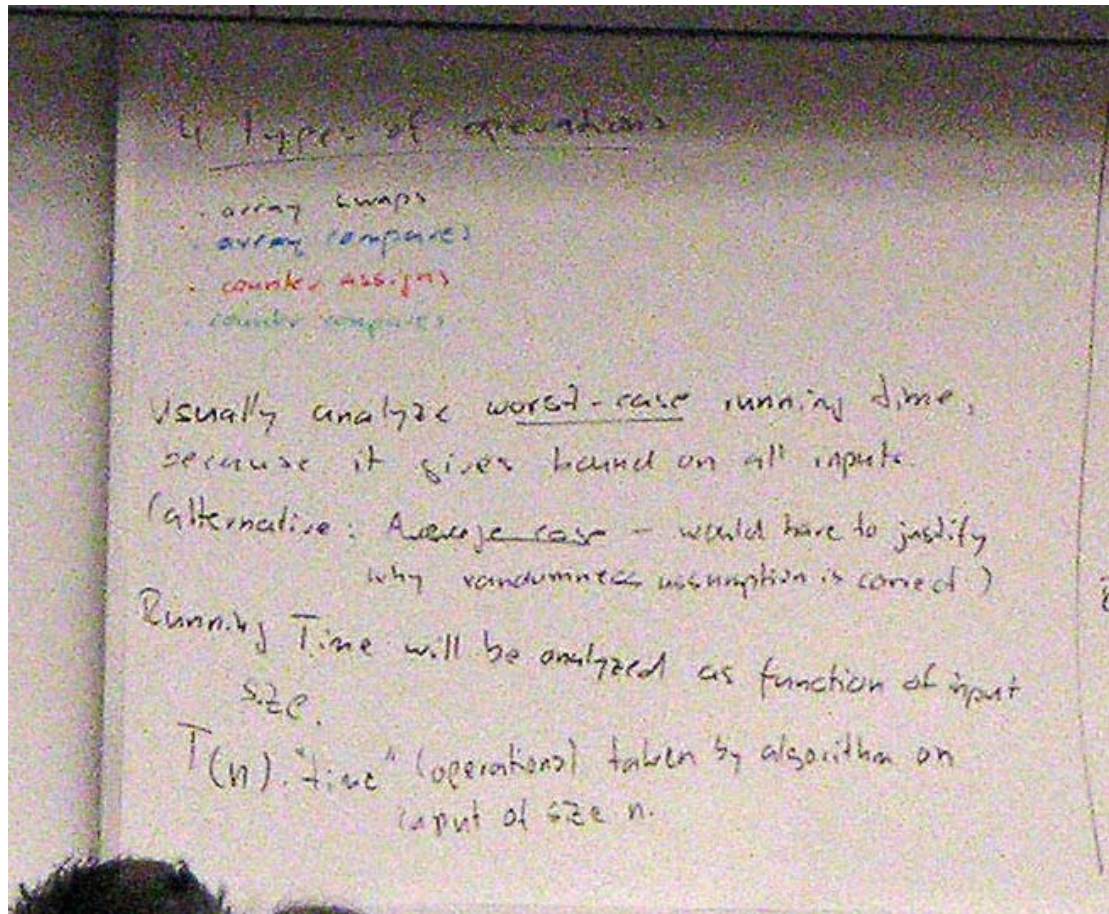
For memory usage, the algorithm just uses the array storage together with 3 extra words.

The running time of the algorithm depends on the array size, how unsorted the array is as well as the computer's calculation speed. Because different instructions and operations may take quite different time, the running time also depends on the relative speed of those operations.

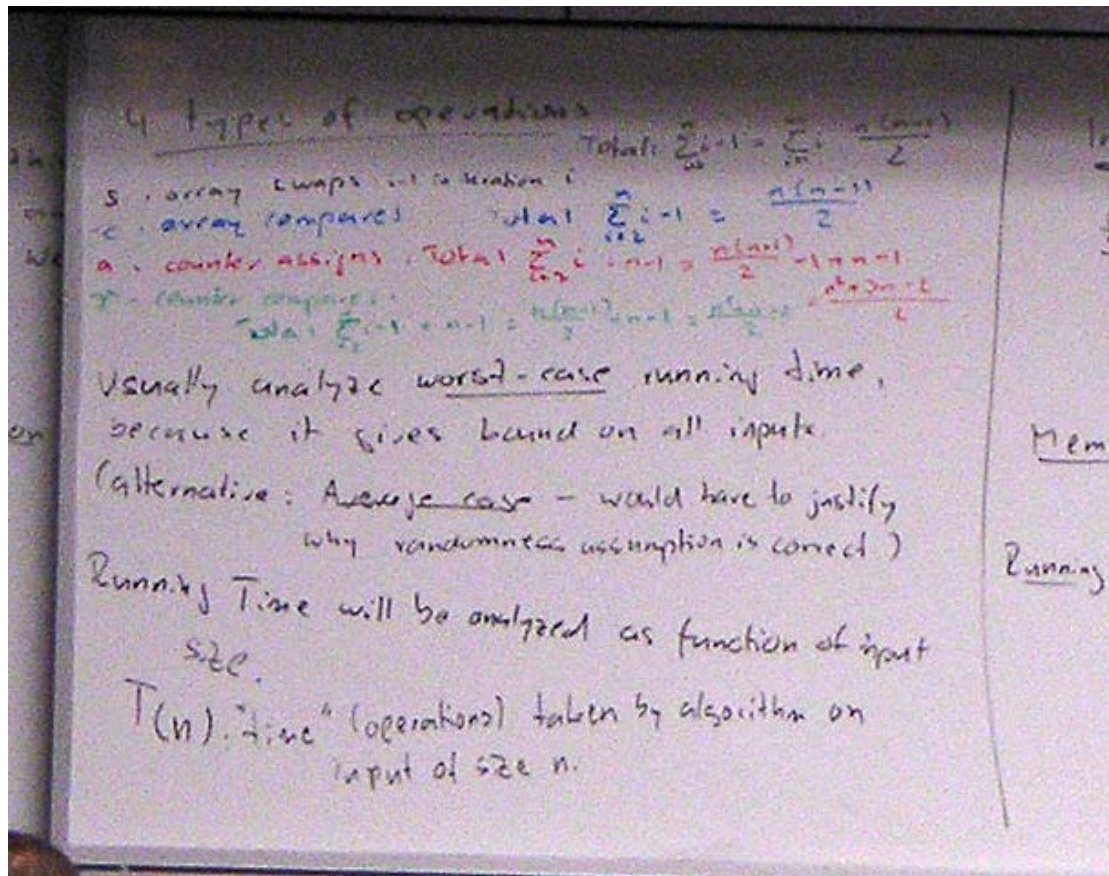


3. Since the running speed depends on so many factors, how can we measure it? Usually the worst-case running time is used because it is always better to tell the users “no matter you feed the program with what kind of inputs, it will take less than some time to calculate it” and the worst-case running time gives the upper bound on all inputs.

The next question is what is the worst-case of Insertion Sort? For the worst-case, first assume that the array is totally unsorted. Then how to measure the array size? It turns out that running time will be expressed as a function of array size, for example:  $T(n)$ , where  $n$  is the size of the input array.

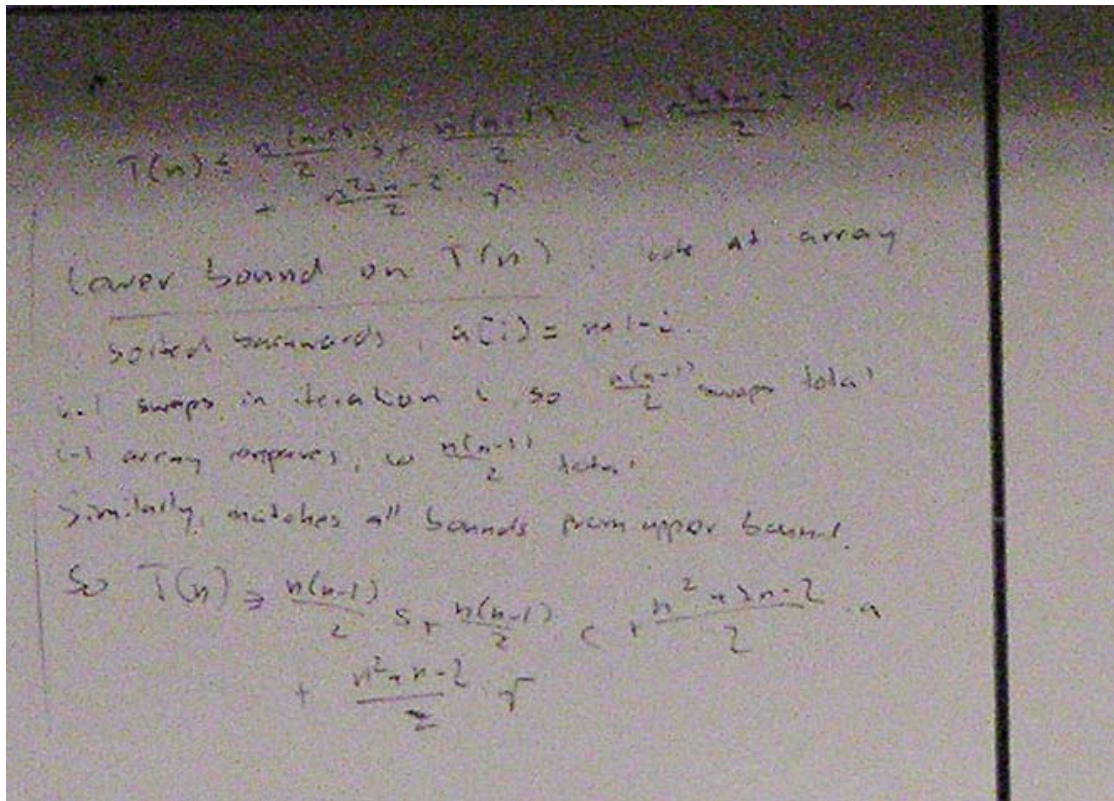


4, Next, the class moved to the detailed worst-case running time analysis of Insertion Sort. For the  $i$ th iteration, in worst-case, the number of swap operation is  $(i-1)$ . So in order to sort the whole array, the total number of swap needed is:  $\sum_{i=1}^n (i-1) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ . The number of array comparison we need to sort the array is identical because there is one comparison before each swap. After similar analysis, we are able to obtain the number of counter assignment and counter comparison we need to sort the array.

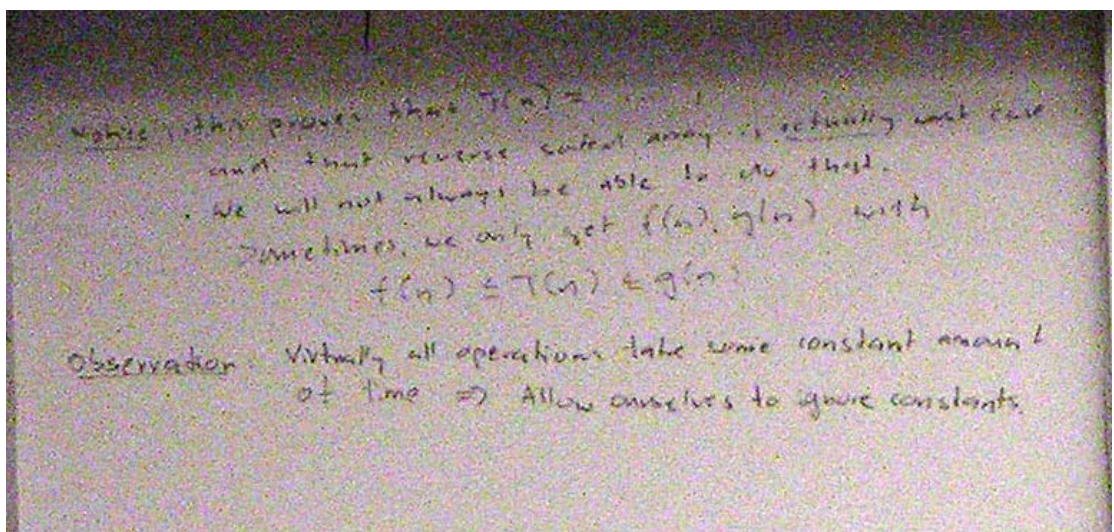


5. To calculate  $T(n)$ , all the four time-consumptions were added up together with respect to the relative speed of those operations.

Afterwards, a specific worst-case array (Inverse Sort) was written onto the blackboard as an example. We can tell that the example is consistent with the formula we derived above.



6, We are not always able to calculate  $T(n)$  directly. Sometimes, we only get  $f(n)$ ,  $g(n)$  with  $f(n) \leq T(n) \leq g(n)$ . Such situation leads to an important concept: Big-O notation (asymptotic notation) which allows us to ignore the constant component.

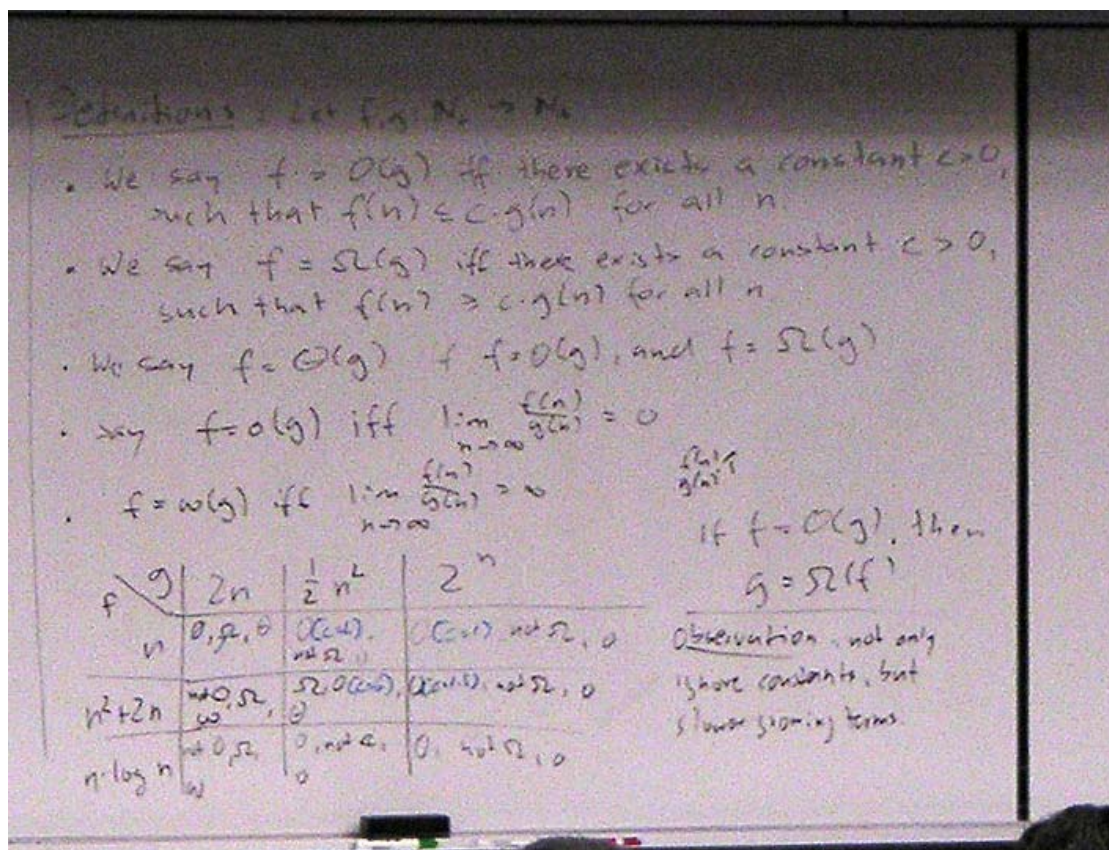


7, The mathematics definition of Big-O notation was given in the lecture. Here, briefly,  $f = O(g)$  means  $f$  grows at most as fast as  $g$ ;  $f = \Omega(g)$  means  $f$  grows at least as fast as  $g$ ;  $f = \theta(g)$  means  $f$  grows nearly the same as  $g$ ;  $f = o(g)$  means  $f$  grows slower than  $g$ ;  $f = \omega(g)$  means  $f$  grows faster than  $g$ .

Since the mathematics definition is very abstract, an example with specific functions was provided. For example, when  $f=n$  and  $g=\frac{1}{2}n^2$ ,  $f = O(g)$  is true when  $c=2$ . From the graph of those two functions, we can see  $\frac{1}{2}n^2$  grows much faster than  $n$  ( $f = o(g)$ ), so  $f = \Omega(g)$  can not be true, neither is  $f = \theta(g)$ .

Through doing the 9 exercises, two good observations are found:

- (1) If  $f = O(g)$ , then  $g = \Omega(f)$ ;
- (2) Big-O notation ignores not only constant (both for addition and multiplication) but also all the terms that grows slower than the main term.



8. With the help of Big-O notation, finally we simplify the worst-case running time of Insertion Sort to  $T(n) = \theta(n^2)$ .

Although Big-O notation is very good for comparing the efficiency of different algorithms, we should pay attention that Big-O notation is not everything. In some application areas, when the scale of the problem is not very large, using a “slightly slower” (measured by Big-O notation) algorithm maybe favorable.

The image shows a handwritten derivation on a piece of paper. It starts with the expression  $\frac{1}{2}n^2 - \frac{1}{2}n = \theta(n^2)$ . Below this, there is a fraction  $\frac{n^2 + 3n - 2}{2} = \theta(n^2)$ . Then, another fraction  $\frac{n^2 + n - 2}{2} = \theta(n^2)$  is written. This is followed by the conclusion  $\therefore T(n) = \theta(n^2)$  and the text "asymptotic running time". At the bottom, two more expressions are written:  $T(n) = 100 \cdot n \log n = \theta(n \log n)$  and  $T'(n) = \frac{1}{2} n^2 = \theta(n^2)$ .

$$\frac{1}{2}n^2 - \frac{1}{2}n = \theta(n^2)$$
$$\frac{n^2 + 3n - 2}{2} = \theta(n^2)$$
$$\frac{n^2 + n - 2}{2} = \theta(n^2)$$

$\therefore T(n) = \theta(n^2)$   
asymptotic running time

$$T(n) = 100 \cdot n \log n = \theta(n \log n)$$
$$T'(n) = \frac{1}{2} n^2 = \theta(n^2)$$