Solution to Exercise 2, Lecture 5

Sept 20, 2001

Exer 2, Lecture 5: Consider the insertion sort algorithm.

void insertionSort(int A[], int n){
    int i, j, k; // Indices used in loops
    newNumber;
    for (k = 1; k < n; k++){ // Insert A[k] into correct slot in A
        newNumber = A[k];
        for (i = 0; A[i] < newNumber; i++) // Find correct place to insert
            ; // note empty body // newNumber into array A

        /* At this point we know that A[0] <= A[1] <= ... <= A[k-1];
          * that all of A[0],...,A[i-1] are < newNumber; that
          * newNumber <= A[i], and that i <= k. So newNumber should go into
          * A[i]. Thus
          * we will move A[i],...,A[k-1] down one spot to make room for
          * newNumber at the ith spot of the array A.
          */

        for (j = k; j > i; j--){ // Make space for newNumber at A[i].
            A[j] = A[j-1];
        }
        A[i] = newNumber; // Put newNumber into its correct spot
    }
}

(a) Explain why the condition (A[i] < newNumber) instead of
    (A[i] < newNumber && i < k) is sufficient in the loop

    for (i = 0; A[i] < newNumber; i++) ; //empty body

(b) Use tcov to study the performance of this algorithm and compare it to
    bubbleSort1. What reasonable conclusions can you come to about which
    algorithm is faster in which situations? Hand-in the tcov output(s) along
    with your analysis.
Answer: (a) Note that $A[i] < \text{newNumber}$ will always be false when $i == k$ since $A[k] == \text{newNumber}$. Thus, the loop will always terminate properly and since the program only has to do only one comparison, namely $A[i] < \text{newNumber}$, the main loop will execute approximately twice as fast compared to doing two comparisons each time. When a value is used to guarantee the termination of a loop, as is done, here, the value is called a sentinel, an often times useful technique for speeding up the execution of loops. Good programmers remain alert for the application of such techniques in their programs.

(b) Insertion sort is one of the fastest algorithms for sorting “small” arrays where small depends on how fast the computer is. To determine the exact point where it becomes faster than other algorithms like quick sort, it is necessary to do empirical studies with the implementations and the computer(s) used. However, this cut-off is usually $\leq 100$ for modern computers. In high-quality, fast sort implementations, insertion sort is often used in conjunction with some other algorithm like quick sort. When the array size is small insertion sort will be used, and when the size is large the other sorting algorithm will be used.