

**Study Module 6 (Analysis of Algorithms)**

This is a simple programming assignment to implement insertion sort algorithm and to observe its worst-case, best-case, and average-case performance. The performance measurement is in terms of the number of key-comparisons, rather than the actual running time.

Implement insertion-sort algorithm without use of recursion. (A recursive implementation of insertion sort for large size  $n$  may cause run-time stack overflow.) To keep track of the number of key-comparisons, it is recommended that the sorting algorithm makes use of a Boolean function  $\text{SMALLER}(A, i, j)$  to do the following:

- Increment a global counter,  $\text{COMPCOUNT}$ , to keep track of the number of key-comparisons performed by the algorithm. (This count is initialized to 0 at the beginning of the algorithm.)
- Perform a comparison between  $A[i]$  and  $A[j]$ . Return  $\text{TRUE}$  if  $A[i] < A[j]$ . Otherwise, return  $\text{FALSE}$ .

Carry out the following experiments.

**1 Small-Size Array,  $n = 32$ .**

Run the algorithm for  $n = 32$  and for each of the following cases:

- Worst-case data input.
- Best-case data input
- Random data input. (Performance on random data represents average-case.)

For each case, print  $n$ , input array, output array (sorted data), and the number of key-comparisons. Does the number of key-comparisons agree with the theoretical values? Theoretically, the worst-case number of key comparisons is  $(n^2 - n)/2$ , and the average number is  $(n^2 - n)/4$ , which is half of the worst-case.

**2 Increasing Array Sizes,  $n = 100$ ,  $n = 1000$ ,  $n = 10000$ .**

Run the algorithm for each of these increasing array sizes and for random data input. For each case, print  $n$  and the resulting number of key-comparisons. (Note that for large  $n$ , it is not practical to print the actual input/output arrays! Also, since the algorithm has  $O(n^2)$  time complexity, an array size larger than 10000 may not be practical.)

Does the number of key-comparisons show  $O(n^2)$  performance? That is, when the array size is increased by a factor of 10, does the number of operations (comparisons) increase by approximately a factor of 100? What is the constant factor for the  $O(n^2)$  performance?

Note: Theoretically, the average number of key-comparisons for insertion sort is  $(n^2 - n)/4$ . Therefore, for large  $n$ , the number of comparisons should be approximately  $n^2/4$ .

Your program must be in C, C++, or JAVA. Submit a hard-copy (paper copy) of your program and the produced output. Also include a short discussion of the results, tabulating the results, and comparing them with the theoretical values.