1- Apply INSERTION-SORT on the array : (code is given)

31,41,59,26,41,58

show the array contents after each iteration.

2- Rewrite the INSERTION-SORT procedure to sort into non-increazing order.

3- Consider the searching problem:

- Input: A sequence of n numbers A = a1, a2, . . . , an

  and a value v.

- Output: An index i such that v = A[i] or the special value NIL if v

  does not appear in A.

Write a suitable pseudocode, which scans through the sequence, looking for v. What is the running time?

4- Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in A[1]. Then find the second smallest element of A, and exchange it with A[2]. Continue in this manner for the first n - 1 elements of A.

- Write pseudocode for this algorithm, which is known as selection sort.

- Discuss the average-case running time.

- Discuss the worst-case running time.

5- For the following array:

31,41,59,26,41,58,33,50,51

A- Write down a pseudo-code for Merge-Sort() algorithm and just explain the main idea of Merge() function.

B- How many levels of recursion for the given array?

C- Show the input to the last called Merge().

D- Show the output of Merge-Sort().

6- **Write a Java code to compare the average-case running time of each of Insertion sort and Merge sort. Hints:**

- Implement both algorithms as user-defined functions in order to call them from your main function.
- The output of each function is the number of comparisons between elements. (The statement of While in insertion sort, and line 13 in Merge() function)
- In your main function: generate 1000 random sequences of integers, each of length 1000, then calculate the average number of comparisons for each algorithm. Comment on the results.