DAA Lab - Session 3 - Bubble Sort and Selection Sort

Brute Force: Implementation of **Bubble Sort** algorithm

Brute Force: Implementation of **Selection Sort** algorithm

**Problem Definition:**

Sort an array of records (a record is a structure with an “id” and a “value” field) using Bubble Sort and Selection Sort in nondecreasing order on the “value” field of the records. Along with the sorted array, return a snapshot of the array after some k number of passes. For Bubble Sort implementation, return the array after k passes when implemented the Bubble Sort algorithm give below. For the Selection Sort implementation, return the array after k passes when implemented the Selection Sort algorithm given below.

**Input:** Input begins with n (1 ≤ n ≤ 220) of number of records indicating the size of the input array. The following n lines has a record per line with an 8-digit id field and a 6-digit value field separated by a space. The following line has an integer k which indicates the number of passes after which the intermediary array must be returned. A zero for k indicates no passes and 1 indicates after the first pass, for instance.

**Output:** For each of Bubble Sort and Selection Sort implementation, return the snapshot of the array after k passes followed by the sorted array. Each record is displayed in a single line with a space between the id and the value field. First n lines are for the snapshot of the array after k-passes during the Bubble Sort, followed by n lines having the sorted array returned by the Bubble Sort. Next n lines are for the snapshot of the array after k-passes during the Selection Sort, followed by n lines having the sorted array returned by the Selection Sort. Finally, the last two lines for the execution time for each algorithm in seconds upto 6 decimal places. If k more than number of passes needed by the algorithm, consider the sorted array as the snapshot after k passes.

**Sample Input:**

5

1 20

2 40

3 20

4 10

5 30

1

**Sample Output:**

1 20

3 20

4 10

5 30

2 40

**4 10**

**1 20**

**3 20**

**5 30**

**2 40**

4 10

2 40

3 20

1 20

5 30

**4 10**

**3 20**

**1 20**

**5 30**

**2 40**

0.000002 sec.

0.000002 sec.

**Algorithms:**

**Algorithm BubbleSort(A[0..n-1])**

//Sorts a given array by Bubble Sort.

//Input: An array A[0..n-1] of orderable elements.

//Output: Array A[0..n-1] sorted in ascending order.

**for *i* ← 0 to n - 2**

**noSwaps ← TRUE**

**for j ← 0 to *n - 2 - i***

***if(A[j] > A[j+1])***

***swap A[j] and A[j+1]***

**noSwaps ← FALSE**

**if (noSwaps = TRUE) return**

**Algorithm SelectionSort(A[0..n-1])**

//Sorts a given array by Selection Sort.

//Input: An array A[0..n-1] of orderable elements.

//Output: Array A[0..n-1] sorted in ascending order.

**for i ← 0 to n-2**

**min ← i**

**for j ← i+1 to n-1**

**if(A[j] < A[min]) min ← j**

**Swap A[i] with A[min]**

**return A**

**Practice-Problems:**

1. In addition, find the number of element-to-element comparisons made in each instance of sorting?
2. Without actually sorting, find the number of passes needed for Bubble Sort algorithm to sort.