NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CS 201–DATA STRUCTURES LAB

Lab Session 07

Instructors: Aqsa Zahid

Objective: In this lab session we will do basic sorting algorithm

This term is used to describe the process of organizing data in a particular order. For example you have an array of 5 numbers [2,4,7,3,1] after sorting it will become [1,2,3,4,7]. We will explore different techniques of sorting, in this lab session we will mainly focus on 3 basic sorting techniques which are as follows:

- Bubble sort
- Selection sort
- Insertion sort

Bubble sort:

The bubble sort algorithm is one of the simplest sorting methods to implement. The way it works is by repeatedly going through the array to be sorted, comparing (and swapping, if in the wrong order) two adjacent elements at a time.

How Bubble Sort Works:

- **I have an array of 5 numbers:**

5	**1**	**12**	**-5**	**16**

- **First iteration:**

5	**1**	**12**	**-5**	**16**

5>1 , swap

1	**5**	**12**	**-5**	**16**

5<12, no action

1	**5**	**12**	**-5**	**16**

12>-5, swap

1	**5**	**-5**	**12**	**16**

12<16, no action

- **Second iteration:**

1	**5**	**-5**	**12**	**16**

1<5, no action

1	**5**	**-5**	**12**	**16**

5>-5, swap

1	**-5**	**5**	**12**	**16**

5<12, no action

1	**-5**	**5**	**12**	**16**

12<16, no action

- **Third iteration:**

1	**-5**	**5**	**12**	**16**

1>-5, swap

-5	**1**	**5**	**12**	**16**

1<5, no action

-5	**1**	**5**	**12**	**16**

5<12, no action

-5	1	5	12	16

12<16, no action

- **Sorted Array:**

-5	1	5	12	16

Algorithm:

```
Void bubblesort (int array[], int n)
{
      for i from 1 to n
       for j from 0 to n - i
          if a[j] > a[j + 1]
             swap( a[j], a[j + 1] )
   }
```

Tasks:

Q1: create a class having function which take an array as argument and sort the array using bubble sort algorithm? Also print each pass or iteration.

Q2: The function bubblesort( ) is inefficient because it continues execution after an array is sorted by performing unnecessary comparisons. Therefore, the number of comparisons in the best and worst case is the same.

The implementation can be improved by making a provision for the case when the array is already sorted.

Modify bubble sort() by adding a flag (Boolean) indicating whether or not it is necessary to make the next pass.

Q3: you can also do sorting in characters i-e A is smaller than B, and B is smaller than C, So do sorting for character array by using bubble sort algorithm and also show its output.

Selection Sort:

It is also one of simplest sorting algorithm. We repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array. Assume that we wish to

sort the array in increasing order, i.e. the smallest element at the beginning of the array and the largest element at the end. We begin by selecting the smallest element and moving it to the first position. We can do this by swapping the smallest element at the first index of array. We then reduce the *effective size* of the array by one element and repeat the process on the smaller (sub)array. The process stops when the effective size of the array becomes 1 (an array of 1 element is already sorted).

How selection sort works:

- Consider an array of size 10

12	23	4	5	0	1	2	7	8	9

- At 0 location we put smallest element by traversing whole array

0	23	4	5	12	1	2	7	8	9

- At 1st location we put another smallest element by traversing sub-array

0	1	4	5	12	23	2	7	8	9

- At 2nd location we put another smallest element by traversing sub-array

0	1	2	5	12	23	4	7	8	9

- At 3rd location we put another smallest element by traversing sub-array

0	1	2	4	12	23	5	7	8	9

- At 4th location we put another smallest element by traversing sub-array

0	1	2	4	5	23	12	7	8	9

- At 5th location we put another smallest element by traversing sub-array

0	1	2	4	5	7	12	23	8	9

- At 6th location we put another smallest element by traversing sub-array

0	1	2	4	5	7	8	23	12	9

- At 7th location we put another smallest element by traversing sub-array

0	1	2	4	5	7	8	9	12	23

Now the array is sorted

Algorithm of selection sort:

```
for out = 1:n,
   min = out
   for in = out+1:n
            if a[min] > a[in]
                        min = in
   swap a[min,out]
end
```

Q1: Rewrite the selection sort program for descending order instead of ascending order.

Insertion sort:

Insertion sort is a simple sorting algorithm, a comparison sort in which the sorted array (or list) is built one entry at a time. An example of an insertion sort occurs in everyday life while playing cards. To sort the cards in your hand you extract a card, shift the remaining cards, and then insert the extracted card in the correct place. This process is repeated until all the cards are in the correct sequence.

How Insertion Sort Works?

We take an unsorted array for our example.

Insertion sort compares the first two elements.

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

And finds that 33 is not in the correct position.

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

These values are not in a sorted order.

So we swap them.

However, swapping makes 27 and 10 unsorted.

Hence, we swap them too.

Again we find 14 and 10 in an unsorted order.

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

This process goes on until all the unsorted values are covered in a sorted sub-list.

- Consider an array of 5 elements

2	4	9	0	6

- On 1st iteration

2	4	9	0	6

- On 2nd iteration

0	2	4	9	6

- On third iteration

0	2	4	6	9

Algorithm of insertion sort:

I

```
INSERTION_SORT (A)
1.    FOR j ← 2 TO length[A]
2.          DO  key ← A[j]
3.                {Put A[j] into the sorted sequence A[1 . . j − 1]}
```

```
4.          i ← j − 1
5.          WHILE i > 0 and A[i] > key
6.                  DO A[i +1] ← A[i]
7.                      i ← i − 1
8.          A[i + 1] ← key
```