JOBSHEET - 6

SORTING (BUBBLE, SELECTION, DAN INSERTION SORT)

6.1 Course Leaning Outcome

After completing this practicum, students should be able to:

a. Create sorting algorithms for bubble sort, selection sort, and insertion sort.

b. Implement sorting algorithms (bubble sort, selection sort, and insertion sort) in a program.

6.2 Experiment 1 - Implementing Sorting Using Objects

Duration : 60 minutes

6.2.1 Experiment Steps

A. SORTING – BUBBLE SORT

1. Create a new folder named **week05**.

2. Create a class **Sorting<Attendance Number>**, then add the following attributes:

```
public class Sorting {
    int[] data;
    int size;
}
```

3. Create a constructor with parameters data[] and jmlDat.

```
public Sorting(int[] data) {
    this.data = data;
    this.size = data.length;
}
```

4. Create a **bubbleSort** method of type void and declare its contents using the Bubble Sort algorithm.

```
public void bubbleSort() {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (data[j] > data[j + 1]) {
                int temp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = temp;
            }
        }
    }
}
```

5. Create a **print** method of type void and declare the content of the method.

```
public void print() {
    for (int num : data) {
        System.out.print(num + " ");
    }
    System.out.println();
}
```

6. Create a class **SortingMain<Attendance Number>**, then declare an array named **a[]** and fill it.

```
int[] a = {34, 7, 23, 32, 5, 62};
```

7. Create a new object named **sorting1**, which is an instantiation of the Sorting class, then fill in its parameters.

```
Sorting sorting = new Sorting(a);
```

8. Call the **bubbleSort** and **print** methods.

```
System.out.println(x:"Original array:");
sorting.print();
sorting.bubbleSort();
System.out.println(x:"Sorted array (Bubble Sort):");
sorting.print();
```

9. Run the program and observe the results.

6.2.2 Verification of Experiment Results

```
Original array:
34 7 23 32 5 62
Sorted array (Bubble Sort):
5 7 23 32 34 62
```

B. SORTING – SELECTION SORT

1. In the **Sorting<Attendance Number>** class created in the previous step, add the **selectionSort** method that implements sorting using the selection sort algorithm.

```
public void selectionSort() {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (data[j] < data[minIndex]) {
                minIndex = j;
            }
        }
        int temp = data[minIndex];
        data[minIndex] = data[i];
        data[i] = temp;
    }
}
```

2. Declare an array named **b[]** in the **SortingMain<Attendance Number>** class, then fill it.

```
int[] b = {30, 20, 2, 8, 14};
```

3. Create a new object named **sorting2**, which is an instantiation of the **Sorting** class, then assign in its parameters.

```
Sorting sorting2 = new Sorting(b);
```

4. Call the **selectionSort** and **print** methods.

```
System.out.println(x:"Original array:");
sorting2.print();
sorting2.selectionSort();
System.out.println(x:"Sorted array (Selection Sort):");
sorting2.print();
```

5. Run the program and observe the results.

6.2.3 Verification of Experiment Results

```
Original array:
30 20 2 8 14
Sorted array (Selection Sort):
2 8 14 20 30
```

C. SORTING – INSERTION SORT

1. In the **Sorting<Attendance Number>** class created in the previous step, add the **insertionSort** method that implements sorting using the insertion sort algorithm.

```
public void insertionSort() {
    for (int i = 1; i < size; i++) {
        int key = data[i];
        int j = i - 1;
        while (j >= 0 && data[j] > key) {
            data[j + 1] = data[j];
            j--;
        }
        data[j + 1] = key;
    }
}
```

2. Declare an array named **c[]** in the **SortingMain<Attendance Number>** class, then fill it.

```
int c[]= {40, 10, 4, 9, 3};
```

3. Create a new object named **dataurut3**, which is an instantiation of the Sorting class, then fill in its parameters.

```
Sorting sorting3 = new Sorting(c);
```

4. Call the **insertionSort** and **display** methods.

```
System.out.println(x:"Original array:");
sorting3.print();
sorting3.insertionSort();
System.out.println(x:"Sorted array (Insertion Sort):");
sorting3.print();
```

5. Run the program and observe the results!

6.2.4 Verification of Experiment Results

```
Original array:
40 10 4 9 3
Sorted array (Insertion Sort):
3 4 9 10 40
```

6.2.5 Questions!

1. Explain the function of the following program code:

```
if (data[j] > data[j + 1]) {
    int temp = data[j];
    data[j] = data[j + 1];
    data[j + 1] = temp;
}
```

2. Show the program code that implements the minimum value search algorithm in selection sort!
3. In insertion sort, explain the purpose of the condition in the loop.

```
while (j >= 0 && data[j] > key)
```

4. In insertion sort, what is the purpose of the given command?

```
data[j + 1] = data[j];
```

6.3 Experiment 2- Sorting Using an Array of Objects

Duration: 45 minutes

6.3.1 Experiment Steps - Sorting Student Data Based on GPA (Bubble Sort)

Observe the **Student** class diagram below! This diagram will serve as a reference for creating the **Student** class.

Student
nim: String name: String studentClass: String gpa: double
Student () Student (nm: String, name: String, clsNm: String, gpa: double) print(): void

The **Student** class will be used to create student objects that will be inserted into an array. It has a default constructor, a parameterized constructor, and a function print() to show all attributes data.

TopStudents
listStudent: Student[5] idx: int
add(std: Student): void print(): void bubbleSort(): void

This class performs operations on the array of student objects, such as adding, displaying, and sorting students using the Bubble Sort algorithm based on GPA.

6.3.2 Experiment Steps

1. Create a class named **Student<Attendance Number>**.
2. Complete the Student class with the following code!

```
public class Student {
    String nim;
    String name;
    String studentClass;
    double gpa;

    public Student() {}

    public Student(String nim, String name, String studentClass, double gpa) {
        this.nim = nim;
        this.name = name;
        this.studentClass = studentClass;
        this.gpa = gpa;
    }

    public void print() {
        System.out.println("NIM: " + nim + ", Name: " + name + ", Class: " + studentClass + ", GPA: " + gpa);
    }
}
```

3. Create a class **TopStudents<Attendance Number>.**

```
public class TopStudent {
    Student[] listStudents;
    int idx;
}
```

4. Add a method **add()** in this class to insert student objects into the **listStudents** attribute.

```
public void add(Student student) {
    if (idx < listStudents.length) {
        listStudents[idx] = student;
        idx++;
    } else {
        System.out.println(x:"List is full!");
    }
}
```

5. Add a method **print()** in this class to display all students.

```
public void print() {
    for (int i = 0; i < idx; i++) {
        listStudents[i].print();
    }
}
```

6. Add a method **bubbleSort()** in this class!

```
public void bubbleSort() {
    for (int i = 0; i < idx - 1; i++) {
        for (int j = 0; j < idx - i - 1; j++) {
            if (listStudents[j].gpa < listStudents[j + 1].gpa) {
                Student temp = listStudents[j];
                listStudents[j] = listStudents[j + 1];
                listStudents[j + 1] = temp;
            }
        }
    }
}
```

7. Create a class **StudentDemo<Attendance Number>**, instantiate a **TopStudents** object, create 5 student objects, and add them using the **add** function. Call **print(),** then sort the data using **bubbleSort(),** and finally call **print()** again.

```
public static void main(String[] args) {
    TopStudent topStudents = new TopStudent(size:5);

    // Adding student data
    topStudents.add(new Student(nim:"2201", name:"Alice", studentClass:"A", gpa:3.9));
    topStudents.add(new Student(nim:"2202", name:"Bob", studentClass:"B", gpa:3.7));
    topStudents.add(new Student(nim:"2203", name:"Charlie", studentClass:"C", gpa:3.8));
    topStudents.add(new Student(nim:"2204", name:"David", studentClass:"D", gpa:3.6));
    topStudents.add(new Student(nim:"2205", name:"Eve", studentClass:"E", gpa:4.0));

    // Display original list
    System.out.println(x:"Original student list:");
    topStudents.print();

    // Sorting students by GPA using Bubble Sort
    topStudents.bubbleSort();
    System.out.println(x:"Sorted student list (by GPA, descending):");
    topStudents.print();
}
```

6.3.3 Verification of Experiment Results

```
Original student list:
NIM: 2201, Name: Alice, Class: A, GPA: 3.9
NIM: 2202, Name: Bob, Class: B, GPA: 3.7
NIM: 2203, Name: Charlie, Class: C, GPA: 3.8
NIM: 2204, Name: David, Class: D, GPA: 3.6
NIM: 2205, Name: Eve, Class: E, GPA: 4.0
Sorted student list (by GPA, descending):
NIM: 2205, Name: Eve, Class: E, GPA: 4.0
NIM: 2201, Name: Alice, Class: A, GPA: 3.9
NIM: 2203, Name: Charlie, Class: C, GPA: 3.8
NIM: 2202, Name: Bob, Class: B, GPA: 3.7
NIM: 2204, Name: David, Class: D, GPA: 3.6
```

6.3.4 Questions!

1. From the following code snippet, answer question a-c:

```
for (int i = 0; i < idx - 1; i++) {
    for (int j = 0; j < idx - i - 1; j++) {
```

   a. Why is the condition in the **bubbleSort()** loop **i < idx - 1**?

   b. Why is the condition in the **bubbleSort()** loop **j < idx - i - 1**?

   c. If the number of data in **listStudents** is 50, how many times will the **i** loop execute? How many stages of Bubble Sort will be performed?

2. Modify the above program to allow dynamic student data input (from the keyboard) consisting of nim, name, studentClass, and gpa.

6.3.5 Sorting Student Data Based on GPA (Selection Sort)

Duration: 30 minutes

If we previously sorted students based on GPA using Bubble Sort in descending order, we will now add a function to sort using Selection Sort in ascending order.

6.3.7 Experiment Steps

1. Modify **TopStudents** to include a **selectionSort()** method.

```
public void selectionSort() {
    for (int i = 0; i < idx - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < idx; j++) {
            if (listStudents[j].gpa < listStudents[minIndex].gpa) {
                minIndex = j;
            }
        }
        // Swap the found minimum element
        Student temp = listStudents[minIndex];
        listStudents[minIndex] = listStudents[i];
        listStudents[i] = temp;
    }
}
```

2. Modify **StudentDemo** and add program lines in **main()** to call **selectionSort()** and **print().**

```
topStudents.selectionSort();
System.out.println(x:"Sorted student list (by GPA, ascending) using Selection Sort:");
topStudents.print();
```

3. Run **StudentDemo** and observe if the data is sorted in ascending order.

6.3.8 Verification of Experiment Results

Make sure the output of your code is like the following image.

```
Sorted student list (by GPA, ascending) using Selection Sort:
NIM: 2204, Name: David, Class: D, GPA: 3.6
NIM: 2202, Name: Bob, Class: B, GPA: 3.7
NIM: 2203, Name: Charlie, Class: C, GPA: 3.8
NIM: 2201, Name: Alice, Class: A, GPA: 3.9
NIM: 2205, Name: Eve, Class: E, GPA: 4.0
```

6.3.9 Questions!

Explain the following code snippet in the correlation with the selection sort!

```
int minIndex = i;
for (int j = i + 1; j < idx; j++) {
    if (listStudents[j].gpa < listStudents[minIndex].gpa) {
        minIndex = j;
    }
}
```

6.3.10 Sorting Student Data Based on GPA Using Insertion Sort

Duration: 30 minutes

6.3.11 Experiment Steps

1. Modify **TopStudents** to include an **insertionSort()** method.

```
public void insertionSort() {
    for (int i = 1; i < idx; i++) {
        Student temp = listStudents[i];
        int j = i;

        // Move elements that have bigger GPA to the right
        while (j > 0 && listStudents[j-1].gpa > temp.gpa) {
            listStudents[j] = listStudents[j-1];
            j--;
        }
        listStudents[j] = temp;
    }
}
```

2. Modify **StudentDemo** and add program lines in **main()** to call **insertionSort()** and **print().**

```
topStudents.insertionSort();
System.out.println(x:"Sorted student list (by GPA, ascending) using Insertion Sort:");
topStudents.print();
```

3. Run **StudentDemo** and observe if the data is sorted in ascending order.

6.3.12 Verification of Experiment Results

Make sure the output of your code is like the following image.

```
Sorted student list (by GPA, ascending) using Insertion Sort:
NIM: 2204, Name: David, Class: D, GPA: 3.6
NIM: 2202, Name: Bob, Class: B, GPA: 3.7
NIM: 2203, Name: Charlie, Class: C, GPA: 3.8
NIM: 2201, Name: Alice, Class: A, GPA: 3.9
NIM: 2205, Name: Eve, Class: E, GPA: 4.0
```

6.3.13 Question

Modify the **insertionSort()** method so that it can perform a descending sorting!

6.4 Assignment

Duration: 45 minutes

Observe the following class diagram and create a program that performs the following menu options:

1. Add Data - Add lecturer data.
2. Display Data - Show all lecturer data.
3. Sort ASC - Sort lecturers by age from youngest to oldest using Bubble Sort.
4. Sort DSC - Sort lecturers by age from oldest to youngest using Selection Sort or Insertion Sort.

Lecturer
id: String name: String gender: Boolean age: int
Lecturer(id: String, name: String, gender: Boolean, age: int) print(): void

LecturerData
lecturerData: Lecturer[10] idx: int
add(dsn: Dosen): void print(): void

sortingASC(): void sortingDSC():void