Lab #4: Sorting Algorithms

The main aim of the lab is to deal with some sorting algorithms and their application.

Deadline: 23:59, 16/10/2023.

==

TASK 1. BASIC SORTING ALGORITHMS

==

Task 1.1: Implement **selection sort** algorithm to sort an array of integers (using **iterative** or **recursive** approach).

```java
// sort by descending order
public static void selectionSort(int[] array) {
     // TODO
     return null;
}
```

Task 1.2: Implement **bubble sort** algorithm to sort an array of integers (using **iterative** or **recursive** approach).

```java
// sort by descending order
public static void bubbleSort(int[] array) {
     // TODO
     return null;
}
```

Task 1.3: Implement **insertion sort** algorithm to sort an array of integers (using **iterative** or **recursive** approach).

```java
// sort by descending order
public static void insertionSort(int[] array) {
     // TODO
     return null;
}
```

Task 1.4: Expand the implemented algorithms to sort an array of order items in Order class **(**see the previous Lab).

==

TASK 2. DIVIDE-AND-CONQUER APPROACH

==

Task 2.1: Implement **merge sort** algorithm to sort an array of integers. The general idea (**Pseudocode**) of the merge sort is described as follows:

```
MergeSort (Array(First..Last))
Begin
If Array contains only one element Then
     Return Array
Else
     Middle= ((Last + First)/2) //rounded down to the nearest
integer
     LeftHalfArray = MergeSort(Array(First..Middle))
     RightHalfArray = MergeSort(Array(Middle+1..Last))
     ResultArray = Merge(LeftHalfArray, RightHalfArray)
     Return ResultArray
EndIf
End MergeSort
```

```
// sort by descending order
public static void mergeSort(int[] array) {
     // TODO
     return null;
}
```

==

Task 2.2: Implement **quick sort** algorithm to sort an array of integers using some strategies for selecting pivot element such as the first element, the last element, a random element and the mean-of-three elements.

The general idea (**Pseudocode**) of the merge sort is described as follows:

```
QuickSort( int[] a )  {
      if ( a.length ≤ 1 )
         return;    // Don't need sorting

      Select a pivot;

      Partition a[] in 2 halves:
         left[]: elements < pivot
          right[]: elements > pivot;

      QuickSort left[];
      QuickSort right[];

      Concatenate: left[] pivot right[]
   }
```

```
// sort by ascending order
public static void quickSort (int[] array) {
     // TODO
     return null;

}

//select pivot element based on the median of three
strategy
private static int getPivot_MedianOfThree(int[]
array) {
     // TODO
     return 0;
}
//  select pivot element based on the first element
in the array
private static int getPivot_First(int[] array) {
     // TODO
     return 0;
}
//  select pivot element based on the last element in
the array
private static int getPivot_Last(int[] array) {
     // TODO
     return 0;
}
//  select pivot element based on choosing a randomly
element in the array
private static int getPivot_Random(int[] array) {
     // TODO
     return 0;

}
```