CPSC 131: Introduction to Computer Programming II

Program 5: Insertion Sort

1 Description of the Program

The insertion sort was discussed and the implementation was demonstrated in the sorting lecture. In this assignment, you are asked to re-implement the insertion sort (`InsertionSort.java`), with additional requirements. Particularly, you need to show:

1. For each iteration, how a number from a unsorted region is placed in the correction posion in the sorted region;
2. How to make the whole array be sorted based on the previous step, and count the total number of shifts during the whole insertion sort process.

2 Details of the Program

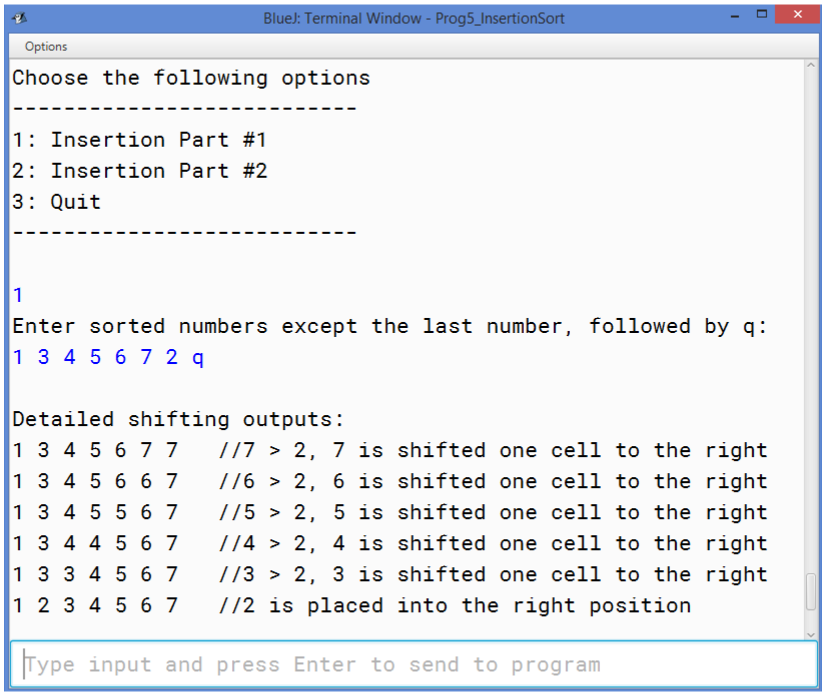
To complete the whole implementation, you should write at least the following important methods:

2.1 Part1: `insertLast`

```
/**
  A method to make an almost sorted array into fully sorted.

  @param arr: an array of integers, with all the numbers are sorted
              excepted the last one
  @param size: the number of elements in an array
*/
public static void insertLast(int[] arr, int size)
{
        // your work
}
```

To make it concrete, let's use the example shown in Figure 1. In this example, the array contains `[1, 3, 4, 5, 6, 7, 2]`, and the size of the array is 7. As we can see, The input array are sorted excepted the last one. The goal here is to place last element into a correct position so all numbers will be sorted. This is exactly what the method `insertLast` is to perform.



```
Choose the following options
---------------------------
1: Insertion Part #1
2: Insertion Part #2
3: Quit
---------------------------


1
Enter sorted numbers except the last number, followed by q:
1 3 4 5 6 7 2 q

Detailed shifting outputs:
1 3 4 5 6 7 7   //7 > 2, 7 is shifted one cell to the right
1 3 4 5 6 6 7   //6 > 2, 6 is shifted one cell to the right
1 3 4 5 5 6 7   //5 > 2, 5 is shifted one cell to the right
1 3 4 4 5 6 7   //4 > 2, 4 is shifted one cell to the right
1 3 3 4 5 6 7   //3 > 2, 3 is shifted one cell to the right
1 2 3 4 5 6 7   //2 is placed into the right position
```

Figure 1: Sampel output for insertion sort part 1.

As we recall from the lecture, the idea is keep shifting all larger number (compared with the last number) to the right until we find the number is smaller than the last number. In our example,

1. Compare 7 with 2. Since $7 > 2$, we copy 7 to its right.
2. Compare 6 with 2. Since $6 > 2$, we copy 6 to its right.
3. Compare 5 with 2. Since $5 > 2$, we copy 5 to its right.
4. Compare 4 with 2. Since $4 > 2$, we copy 4 to its right.
5. Compare 3 with 2. Since $3 > 2$, we copy 3 to its right.
6. Compare 1 with 2. Since $1 < 2$, we stop here and copy 2 to its current position.

In this method, you are **required** to print the following information for each iteration until the termination of the loop (See Figure 1):

- The intermediate array content and
- The comparing pair information

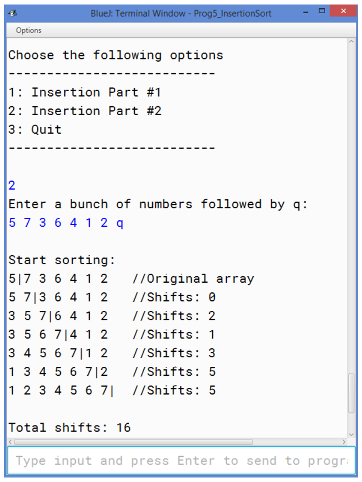
In this example, five numbers (3, 4, 5, 6, and 7) are greater than 2, so you have five (5) shifts, and five lines of outputs, plus one line putting 2 in its right position.

2.2 Part2: `insertionSort`

```
/**
  A method to make an unsorted array into fully sorted.

  @param arr: an array of integers
  @param size: the number of elements in an array
*/
public static void insertionSort(int[] arr, int size)
{
        // your work
}
```

This method is to make an unsorted array into sorted, which we have discussed in the lecture. The basic idea is to run $n-1$ iterations, and for each iteration, we perform the task as we implemented in Part 1.

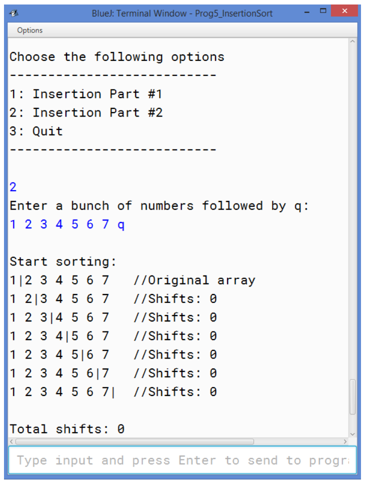

BlueJ: Terminal Window - Prog5_InsertionSort
Options
Type input and press Enter to send to progr

```
Choose the following options
--------------------------
1: Insertion Part #1
2: Insertion Part #2
3: Quit
--------------------------

2
Enter a bunch of numbers followed by q:
5 7 3 6 4 1 2 q

Start sorting:
5|7 3 6 4 1 2   //Original array
5 7|3 6 4 1 2   //Shifts: 0
3 5 7|6 4 1 2   //Shifts: 2
3 5 6 7|4 1 2   //Shifts: 1
3 4 5 6 7|1 2   //Shifts: 3
1 3 4 5 6 7|2   //Shifts: 5
1 2 3 4 5 6 7|  //Shifts: 5

Total shifts: 16
```

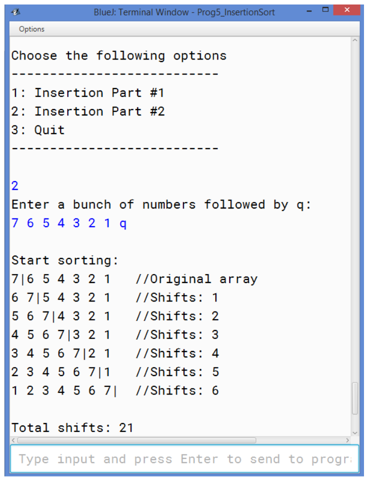

BlueJ: Terminal Window - Prog5_InsertionSort
Options
Type input and press Enter to send to progr

```
Choose the following options
--------------------------
1: Insertion Part #1
2: Insertion Part #2
3: Quit
--------------------------

2
Enter a bunch of numbers followed by q:
1 2 3 4 5 6 7 q

Start sorting:
1|2 3 4 5 6 7   //Original array
1 2|3 4 5 6 7   //Shifts: 0
1 2 3|4 5 6 7   //Shifts: 0
1 2 3 4|5 6 7   //Shifts: 0
1 2 3 4 5|6 7   //Shifts: 0
1 2 3 4 5 6|7   //Shifts: 0
1 2 3 4 5 6 7|  //Shifts: 0

Total shifts: 0
```


BlueJ: Terminal Window - Prog5_InsertionSort
Options
Type input and press Enter to send to progr

```
Choose the following options
--------------------------
1: Insertion Part #1
2: Insertion Part #2
3: Quit
--------------------------

2
Enter a bunch of numbers followed by q:
7 6 5 4 3 2 1 q

Start sorting:
7|6 5 4 3 2 1   //Original array
6 7|5 4 3 2 1   //Shifts: 1
5 6 7|4 3 2 1   //Shifts: 2
4 5 6 7|3 2 1   //Shifts: 3
3 4 5 6 7|2 1   //Shifts: 4
2 3 4 5 6 7|1   //Shifts: 5
1 2 3 4 5 6 7|  //Shifts: 6

Total shifts: 21
```

Figure 2: Outputs for part 2: average (Left), best (Middle), and worst (Right) cases

To make it concrete, let's use the example shown in Figure 2 (Left). In this example, the array contains `[5, 7, 3, 6, 4, 1, 2]`, and the size of the whole array is 7. Since the size of the array $n = 7$, you need 6 iterations to get the whole array to be sorted. Particularly,

1. Iteration #1: Perform `insertLast` on subarray `[5, 7]`
2. Iteration #2: Perform `insertLast` on subarray `[5, 7, 3]`
3. Iteration #3: Perform `insertLast` on subarray `[3, 5, 7, 6]`

4. Iteration #4: Perform `insertLast` on subarray `[3, 5, 6, 7, 4]`

5. Iteration #5: Perform `insertLast` on subarray `[3, 4, 5, 6, 7, 1]`

6. Iteration #6: Perform `insertLast` on subarray `[1, 3, 4, 5, 6, 7, 2]`

In this method, you are **required** to print the following information (1) The intermediate array content and (2) The numbers of shift for each iteration, and the total number of shifts (See Figure 2).

Important: Since the `insertLast` prints out intermediate results, you may NOT directly use this method in `insertionSort`. You could copy the original source code from `insertLast` and disable the printing statements. or write another `insertLast` method without printing statements.

2.3 Addtional: `printArray`

`public static void printArray(int [] arr, int size)`

A variety of `printArray` methods with different parameters (designed on your own) maybe useful as you will print out array each iteration used in previous methods.

2.4 `main()`

To test your program, you may provide a menu list for user to choose (See Figures 1 and 2), declare an array with the size (given from the keyboard) or fixed number, fill up the array values from the keyboard, invoke one of these two methods (`insertLast` and `insertionSort`), and print out the **required formatted program outputs** (See Figures 1 and 2).

3 Additional Requirements

1. Your program should not terminate if user does not enter 3 (means terminating the program). (**Hint**: you may use `do-while` loop to control it in your `main()`).

2. For the testing of part1, make sure you enter **an input array with almost sorted**, *i.e.*, all elements in the array are sorted except the last one.

3. For the testing of part 2, you should test your program with the inputs of (a) **unsorted array** (average case), (b) **sorted array in ascending order** (best case), and (c) **sorted array in descending order** (worst case). Save screenshots for each case like I did in Figure 2.

4. Your program should have good style (indentation, whitespace, comments, vertical alignment, ...).

```
//System.out.printf("\t\t//%d > %d, %d is shifted one cell to the right\n", arr[j], next, arr[j]);
        //System.out.printf("\t\t//%d is placed into the right position\n", next);
```

4 Submission

Upload the following items on D2L dropbox, including:

1. The source code (`InsertionSort.java`);
2. Output screenshot for insertion part 1 (Like Figure 1);
3. Output screenshots for **three** cases for part 2 (Like Figure 2).