CSC 1351-3, Spring 2020, Lab 5 Name this project SortTimer2

Create a class SortTimer2 with the following Methods:

1. **public static void selectionSort(int[] nums)**
   This method uses the selection sort to sort an array of integers, named nums, into increasing order.

2. **public static void selectionSortPrint(int[] nums)**
   Create this method by copying and pasting the selectionSort method in Step 1 above and then adding the statement:
   `System.out.println("nums = " + Arrays.toString(nums));`
   at the end of the first for loop (right before the } that ends the first for loop).

3. **public static void insertionSort(int[] nums)**
   This method uses the insertion sort to sort an array of integers, named nums, into increasing order.

4. **public static void insertionSortPrint(int[] nums)**
   Create this method by copying and pasting the insertionSort method in Step 3 above and then adding the statement:
   `System.out.println("nums = " + Arrays.toString(nums));`
   at the end of the first for loop (right before the } that ends the first for loop).

5. **public static void mergeSort(int[] nums, int left, int right)**
   This method sorts the subarray nums[left..right] of the nums array into increasing order using the merge sort algorithm. Use the following pseudocode for this method:

   If left < right, do the following:
   a. Let mid = (left + right)/2.
   b. If left ≤ mid, make a recursive call mergeSort(nums, left, mid) to sort the subarray nums[left..mid].
   c. If mid+1 < right, make a recursive call mergeSort(nums, mid+1, right) to sort the subarray nums[mid+1..right].
   d. Create a new integer array result[] of size right-left+1 by merging the sorted subarrays nums[left..mid] and nums[mid+1..right] as follows:
      d1. Initialize m = left, n = mid+1, k = 0.
      d2. While ((m ≤ mid) && (n ≤ right)), copy the smaller of nums[m] and nums[n] to result[k] and then increase k by 1 and increase by 1 either m or n depending on whether nums[m] or nums[n] was copied.
      d3. If (n > mid+1), then if (m ≤ mid) copy the remainder of nums[left..mid] into the result array, otherwise copy the remainder of nums[mid+1..right] into the result array. Then copy the result array to the subarray nums[left..right].

6. **public static void mergeSortPrint(int[] nums, int left, int right)**
   Create this method by copying and pasting the mergeSort method in Step 5 above and then adding the statement:
   `System.out.println("nums = " + Arrays.toString(nums));`
   at the end of the method (right before the } at the end of method).

7. **public static void quickSort(int[] nums, int left, int right)**
This method sorts the subarray nums[left..right] of the nums array into increasing order using the quick sort algorithm. Use the following pseudocode for this method:

If left < right, do the following:
   a. Call the partition method in Step 8 below to get the partition index p.
   b. If left < p, make a recursive call quickSort(nums, left, p) to sort the subarray nums[left..p].
   c. If p+1 < right, make a recursive call quickSort(nums, p+1, right) to sort the subarray nums[p+1..right].

8. **public static int partition(int[] nums, int left, int right)**
This method partitions the subarray nums[left..right] of the nums array into two smaller subarrays. Use the following pseudocode for this method:

a. Let pivot = nums[left].
b. Initialize i = left-1 and j = right+1 .
c. While (i < j), do the following:
   c1. Keep on incrementing i until nums[i] >= pivot .
   c2. Keep on decrementing j until nums[j] <= pivot .
   c3. If i < j, interchange nums[i] and nums[j] .
d. Return j .

9. **public static void quickSortPrint(int[] nums, int left, int right)**
Create this method by copying and pasting the quickSort method in Step 7 above and then adding the statement:
```
System.out.println("nums = " + Arrays.toString(nums));
```
right after the call of the partition method.

10. **public static void main(String[] args)**

The first part of the output of this program, which shows the array after each pass of the 4 sort methods when a small array is being sorted, must be exactly the same as the output shown below.

The second part of the output of this program, which shows the sort times, should be similar to the sample output shown below (the actual sort times will vary from one run of this program to the next due to variation in the computer's execution time).

First Part of the Output

```
Selection Sort:
nums = [5, 6, 4, 7, 2, 1, 8, 3]
nums = [1, 6, 4, 7, 2, 5, 8, 3]
nums = [1, 2, 4, 7, 6, 5, 8, 3]
nums = [1, 2, 3, 7, 6, 5, 8, 4]
nums = [1, 2, 3, 4, 6, 5, 8, 7]
nums = [1, 2, 3, 4, 5, 6, 8, 7]
nums = [1, 2, 3, 4, 5, 6, 8, 7]
nums = [1, 2, 3, 4, 5, 6, 7, 8]

Insertion Sort:
nums = [5, 8, 4, 6, 2, 1, 7, 3]
nums = [5, 8, 4, 6, 2, 1, 7, 3]
nums = [4, 5, 8, 6, 2, 1, 7, 3]
nums = [4, 5, 6, 8, 2, 1, 7, 3]
nums = [2, 4, 5, 6, 8, 1, 7, 3]
nums = [1, 2, 4, 5, 6, 8, 7, 3]
nums = [1, 2, 4, 5, 6, 7, 8, 3]
nums = [1, 2, 3, 4, 5, 6, 7, 8]

Merge Sort:
nums = [5, 3, 6, 4, 2, 0, 1]
nums = [3, 5, 6, 4, 2, 0, 1]
nums = [3, 5, 4, 6, 2, 0, 1]
nums = [3, 4, 5, 6, 2, 0, 1]
nums = [3, 4, 5, 6, 0, 2, 1]
nums = [3, 4, 5, 6, 0, 1, 2]
nums = [0, 1, 2, 3, 4, 5, 6]

Quick Sort:
nums = [5, 8, 4, 7, 2, 1, 6, 3]
nums = [3, 1, 4, 2, 7, 8, 6, 5]
nums = [2, 1, 4, 3, 7, 8, 6, 5]
nums = [1, 2, 4, 3, 7, 8, 6, 5]
nums = [1, 2, 3, 4, 7, 8, 6, 5]
nums = [1, 2, 3, 4, 5, 6, 8, 7]
nums = [1, 2, 3, 4, 5, 6, 8, 7]
nums = [1, 2, 3, 4, 5, 6, 7, 8]
```

Sample Output for the Second Part of the Output

```
Enter the starting value for the length n of the array to be sorted, the stepsize by which n is
increased, and the number of steps:
20000 20000 3

n= 20000  Sort Run Time (milliseconds): Selection:  107   Insertion:  35   Merge:  2   Quick: 0
n= 40000  Sort Run Time (milliseconds): Selection:  328   Insertion: 149   Merge:  2   Quick: 2
n= 60000  Sort Run Time (milliseconds): Selection:  727   Insertion: 339   Merge:  6   Quick: 2
n= 80000  Sort Run Time (milliseconds): Selection: 1341   Insertion: 600   Merge: 11   Quick: 4
```