Assignment Prefix: lab12

Points: 100

Due Date: Wednesday, April 21, 2017 @ 11:59pm

For this assignment you are going to implement several sorting algorithms.

RESTRICTIONS:

* You may **NOT** import **java.util.Comparator**
  + you must write your own comparators
* You may **NOT** import **java.util.Arrays**
  + You must write your own copy methods for any arrays.
  + You must write your own print methods for any arrays.
  + You may **NOT** use any of the Java Array sorting features.
  + **Exception, in the mergeSort you can call the Arrays.copyOfRange method**
* You may **NOT** import any other Java container class.
  + e.g. you must use your own Queue, LinkedQueue, SinglyLinkedList, etc. classes.

**Task 1:**

Create a Voter Class that encapsulates the concept of a voter. The attributes of a voter are:

* id
  + a random integer in the range 0 to 99999999 (i.e. like a social security number)
  + we will ignore the fact that we may get duplicate id numbers
* name
  + a String of a random length between 5 and 10 characters (inclusive) made up of a random set of lower case characters
* party
  + must be one of the following (distributed as indicated):
    - democratic 30%
    - republican 30%
    - independent 20%
    - other 20%
* voted
  + must be one of the following (distributed as indicated):
    - yes 58%
    - no 42%

**Task 2:**

Create a class named Sort that will act as a container for the following generic array sorting algorithms:

* simpleBubbleSort
  + a brute force bubble sort that just uses a pair of nested loops
  + this needs to be a generic bubble sort
  + this needs to be a stable sort
* insertionSort
  + an iterative sort described on pages 524 and 525 of the Java Illuminated Text
  + modify this code to make it a generic sort
* selectionSort
  + an iterative sort described on pages 524 and 525 of the Java Illuminated Text
  + modify this code to make it a generic sort
* mergeSort
  + this should be the recursive mergeSort described in the textbook
* quickSort
  + this should be the recursive quickSort described in the textbook
  + you may have to modify this code
* radixSort
  + as described in lecture, not as described in the textbook
  + the radixSort should be able to support between two and four keys
  + the first parameter in the parameter list should be the array being sorted.
  + the remaining parameters in the parameter list should be the keys, ordered left to right from most significant to least significant

Task 3

* Create concrete comparators for each of the attributes of the Employee class.

**Task 4:**

* Create a client class that
  + Prints the run times for each of the following sorts
  + Each sort should be in ascending order for its key value
  + Generates an array of 1,000,000 voters
  + Sort the voter array on name using the merge sort
  + Sort the voter array on party using the quick sort
  + **For bubble, insertion and selection sorts**
    - **Reduce the array size to 100,000 for the bubble sort.**
    - **A list of 1,000,000 voters can take 6 plus hours to sort with the bubble sort.**
  + Sort the employee array on id using the bubble sort
  + Sort the voter array on voted using the insertion sort
  + Sort the voter array on party using the selection sort
  + Sort the voter array using the radix sort so that
    - All voters are sorted by party
    - Within a party grouping all the voters are sorted by voted
    - Within a party and voted grouping all the voters are sorted by their name
* Since the list of voter is long
  + You will not print out the unsorted or sorted voter lists, instead,
  + ***Print out the time that it takes to run each sort***
  + Suggestion:
    - Make a test run of 1,000 employees and inspect the results to make sure that they are correctly ordered but you should not display them in your Word document
* Caution
  + Make sure that you are passing the same unsorted list to each of your sort routines.
  + If you follow the textbook code, the container that is passed in as a parameter is the container that is sorted.

**Turning in your assignment:**

* **Make sure that all of your code is properly documented.**
* Turn in your assignment using the standard method.
* Create a Word document using the standard naming convention:
  + Lab11-LnameF.docx
* Copy and paste each of your Java files into the document.
  + You only need to include the Java files specific to this assignment
    - Comparator files
    - Sort class
    - Client class
    - Any other files written specifically for this assignment.
* Paste the screenshots showing the complete output of a complete run of your program after the Java code in your document.
* Export your NetBeans project to a zip archive.
* Turn in the Word document and zipped project as to separate files in a single Blackboard submission.