Points: 100

Due Date: Tuesday, April 23, 2019 @ 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 may **NOT** use any of the Java Array sorting features.
  + You may NOT use any of the Java Array copying features.
    - Exception, the textbook code for the mergeSort uses the Arrays.copyOfRange method. You use this method but only in the mergeSort method.
* You may **NOT** import any other Java container class.
  + e.g. you must use your own Queue class.

**Task 1:**

Create an employee Class that encapsulates the concept of an employee. The attributes of an employee 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
* dept
  + a random integer in the range 1 to 5 (inclusive)
* hired
  + a random integer in the range 2008 to 2018 (inclusive)

**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
* enhancedBubbleSort
  + a bubble sort that only sorts the unsorted portion of the array on each pass through the inner loop
  + a bubble sort that stops as soon as it knows the list is sorted
  + this needs to be a generic sort
  + this needs to be a stable sort
* insertionSort
  + the insertion sort as discussed in class
  + you may use the code from the Java Illuminated text modified to be generic
  + make sure it is a stable sort
* selectionSort
  + the insertion sort as discussed in class
  + you may use the code from the Java Illuminated text modified to be generic
  + make sure it is a stable sort
* mergeSort
  + this must be one of the mergeSort methods described in the textbook
  + make sure it is a stable sort
  + you may have to modify this code
* quickSort
  + this must be the quickSort methods described in the textbook
  + make sure it is a stable sort
  + you may have to modify this code
* radixSort
  + this should be the radix sort describe in class
  + this should be a generic sort
  + 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
  + you should use the Radix sort described in class and not the bucket approach described in the textbook

**Task 3:**

* Create a client class that tests your sorting algorithms for value of N (number employees in the list) where N goes from 1,000 to 1,000,000,000 increasing by a multiple of 10 each time. i.e. N goes from 1,000 to 10,000, to 100,000 to 1,000,000…
* For each value of N time the following operations
  + Generate an array of N employees
  + Sort the employee array on name using the merge sort
  + Sort the employee array on dept using the quick sort
  + Sort the employee array on id using the bubble sort
  + Sort the employee array on hired using the enhanced bubble sort.
  + Sort the employee array on name using the insertion sort
  + Sort the employee array on id using the selection sort
  + Sort the employee array using the radix sort so that
    - All employees are sorted by department
    - Within a department grouping all the employees are sorted by hire date
    - Within a department and hire date grouping all the employees are sorted by their name
* Since the list of employees is long
  + You will not print out the unsorted or sorted employee lists, instead,
  + Print out the time that it takes to run each sort
  + Suggestion:
    - Make a test run of 100 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.
    - Do this by making a copy of your initial list and passing the copy to the sort method.
    - Make the copy before a call to a sort method.
    - Your client should never have more than one copy of the initial list in memory at any given time.
  + Each of your sort methods should sort the copy of the initial list.
  + For each sort make sure that the sorted list is deleted before moving on to the next sort method.
  + At any given time, you should have at most two list
    - The initial unsorted list for a specific value of N
    - The copy of the initial list in either an unsorted or sorted state.
* You must include a timedOut feature in your code.
  + If any major section of your code exceeds the timeOut value then that section should stop executing and your program should, when possible, move on to the next section.
  + Major sections of your code are defined as
    - Creating the initial unsorted list
    - Making a copy of the unsorted list
    - Each of the sort routines
  + Before turning in the assignment set the timeOut value to 60,000 msec (i.e. 10 minutes).
* Your program must be robust enough to recognize and handle an OutOfMemoryError.
  + If any major section of your code exceeds the available memory that section should stop executing and your program should, when possible, move on to the next section.
  + Major sections of your code are defined above.

It is suggested that you print out the unsorted and sorted lists for some small value of N to verify that your sort methods are working correctly.

Do NOT turn in any printed lists with your assignment.

You must display the results of your test in an ASCII table. This ASSCII table may look something like:

+-------------------------------------------------------------------------------------------+

| Run time in milliseconds |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

| N | create | copy | bubble | eBubble | Selection | Merge | Radix |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

| 1,000 | 10 | 0 | 25 | 21 | 33 | 20 | 45 |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

| 10,000 | 31 | 0 | 891 | 752 | 650 | 15 | 55 |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

| 100,000 | 156 | 0 | timeOut | timeOut | timeOut | 78 | 213 |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

| 1,000,000 | 1,332 | 0 | timeOut | timeOut | timeOut | 240 | 720 |

+-------------+----------+--------+-----------+-----------+-------------+---------+---------+

* Not all columns or rows are included in the above example.
* Your table should go from 1,000 to 1,000,000,000 increasing N by a multiple of 10 each time.
* Your table will probably need to include OutOfMemory notations in appropriate cells.
* You can make up your own column header names and timeOut and OutOfMemory messages.

**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 and 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.