CS 46B

Introduction to Data Structures

Summer Semester 2015

Department of Computer Science
San José State University
Instructor: Ron Mak

Homework #7
Quicksort

Assigned:	Monday, July 6
Draft due:	Wednesday, July 8 at 11:59 PM
Partition Codecheck URL:	http://codecheck.it/codecheck/files/15070404339sixelvygopt9syaqspx77ror
Quicksort Codecheck URL:	http://codecheck.it/codecheck/files/1507040626wmpa3ewx8lpf4g77v8jbc372
Canvas:	Homework 7 Draft
Points:	10 points max
Final due:	Friday, July 10 at 11:59 PM
Presidents Codecheck URL:	http://codecheck.it/codecheck/files/1507040517919iude2tuhn0jfio6q8k87xr
Canvas:	Homework 7 Final
Points:	20 points max

This assignment allows you to study the famous quicksort algorithm and its partitioning method, and it give you practice using quicksort to sort a list of names.

Partitioning

The partitioning algorithm in the textbook chooses the first element of an array range to be the pivot element. It partitions the range into two subranges, where all the elements in the first subrange are less than the elements in the second subrange. Recursively, quicksort then sorts each subrange.

A slightly more elegant version of the partitioning algorithm, as described in class, chooses a pivot element and partitions an array range by creating a subrange of all the values less than the pivot value and the other subrange of all the values greater than the pivot value. It places the pivot element in its "final resting place" to separate the two subranges. The pivot element will not move during all subsequent operations to sort the array. Otherwise, this version of the partitioning algorithm works like the one in the textbook – index variables `i` and `j` march towards each other and along the way, certain elements of the array range are swapped.

Here's an example of a trace of the execution of the partitioning algorithm:

```
Partitioning:
    [50, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 88]
Pivot = 50
    [88, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 50]
i = 0, j = 18, swapped 4 and 88:
    [4, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 88, 50]
i = 1, j = 16, swapped 32 and 63:
    [4, 32, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 63, 54, 88, 50]
i = 4, j = 13, swapped 18 and 86:
    [4, 32, 29, 14, 18, 16, 79, 16, 26, 61, 47, 64, 83, 86, 97, 92, 63, 54, 88, 50]
i = 6, j = 10, swapped 47 and 79:
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 61, 79, 64, 83, 86, 97, 92, 63, 54, 88, 50]
Partitioned: pivot = 50, pivot index = 9
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
```

The algorithm first selects the first element of the range, 50, as the pivot. It moves the pivot "out of the way" by swapping it with the last element of the range.

Index variable `i` is set to the first element of the range, and index variable `j` is set to the second-to-the-last element of the range. (The pivot temporarily resides in the last element.)

As shown in the trace output, `i` marches to the right as long as the `i`th element value is less than the pivot value, and `j` marches to the left as long as the `j`th element value is greater than the pivot value. Whenever `i` and `j` both have to stop, the `i`th and `j`th elements are swapped. Then `i` and `j` resume their marches toward each other with more swaps whenever necessary. When `i` and `j` cross each other, `i` is the index of the pivot element. The pivot value is swapped in from its temporary location to its final position indexed by `i`. In the trace above, pivot value 50 ends up in the 9th position; all elements to the left are less than the pivot, and all elements to the right are greater than the pivot.

Draft

The above trace of the partitioning algorithm is printed by program **PartitionTracer.java**. However, the version you are provided in Codecheck only writes the initial array range and the final partitioned range:

```
Partitioning:
    [50, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 88]
Partitioned:
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
```

Modify class `PartitionTracer` to print the trace output shown above to `System.out`.

File **partitiontraceroutput.txt**:
http://www.cs.sjsu.edu/~mak/CS46B/assignments/7/partitiontracer.out

Draft Partition Codecheck URL:
http://codecheck.it/codecheck/files/15070404339sixelvygopt9syaqspx77ror

To see how the partitioning algorithm works with quicksort, we can also trace recursive calls to sort. Note how once a pivot element has been swapped into its final position, it does not move during the rest of the sorting operations:

```
Original array:
    [50, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 88]

SORTING: from = 0, to = 19
Partitioning:
    [50, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 88]
Pivot = 50
    [88, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 4, 50]
i = 0, j = 18, swapped 4 and 88:
    [4, 63, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 32, 54, 88, 50]
i = 1, j = 16, swapped 32 and 63:
    [4, 32, 29, 14, 86, 16, 79, 16, 26, 61, 47, 64, 83, 18, 97, 92, 63, 54, 88, 50]
i = 4, j = 13, swapped 18 and 86:
    [4, 32, 29, 14, 18, 16, 79, 16, 26, 61, 47, 64, 83, 86, 97, 92, 63, 54, 88, 50]
i = 6, j = 10, swapped 47 and 79:
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 61, 79, 64, 83, 86, 97, 92, 63, 54, 88, 50]
Partitioned: pivot = 50, pivot index = 9
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]

SORTING: from = 0, to = 8
Partitioning:
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Pivot = 4
    [26, 32, 29, 14, 18, 16, 47, 16, 4, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Partitioned: pivot = 4, pivot index = 0
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]

SORTING: from = 0, to = -1

SORTING: from = 1, to = 8
Partitioning:
    [4, 32, 29, 14, 18, 16, 47, 16, 26, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Pivot = 32
    [4, 26, 29, 14, 18, 16, 47, 16, 32, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
i = 6, j = 7, swapped 16 and 47:
    [4, 26, 29, 14, 18, 16, 16, 47, 32, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Partitioned: pivot = 32, pivot index = 7
    [4, 26, 29, 14, 18, 16, 16, 32, 47, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]

SORTING: from = 1, to = 6
Partitioning:
    [4, 26, 29, 14, 18, 16, 16, 32, 47, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Pivot = 26
    [4, 16, 29, 14, 18, 16, 26, 32, 47, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
i = 2, j = 5, swapped 16 and 29:
    [4, 16, 16, 14, 18, 29, 26, 32, 47, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]
Partitioned: pivot = 26, pivot index = 5
    [4, 16, 16, 14, 18, 26, 29, 32, 47, 50, 79, 64, 83, 86, 97, 92, 63, 54, 88, 61]

SORTING: from = 1, to = 4

 • • •

SORTING: from = 19, to = 19

Sorted array:
    [4, 14, 16, 16, 18, 26, 29, 32, 47, 50, 54, 61, 63, 64, 79, 83, 86, 88, 92, 97]
```

Modify program **QuicksortTracer.java** provided in Codecheck to write the trace output.

File **sorttraceroutput.txt**:
http://www.cs.sjsu.edu/~mak/CS46B/assignments/7/sorttracer.out

Draft Quicksort Codecheck URL**:**
http://codecheck.it/codecheck/files/1507040626wmpa3ewx8lpf4g77v8jbc372

Submit both signed zip files in Canvas: **Homework 7 Draft**.
Due: Wednesday, July 8 at 11:59 PM

Final

```
George Washington
John Adams
Thomas Jefferson
James Madison
James Monroe
John Quincy Adams
Andrew Jackson
Martin Van Buren
William Henry Harrison
John Tyler
James K. Polk
Zachary Taylor
Millard Fillmore
Franklin Pierce
James Buchanan
Abraham Lincoln
Andrew Johnson
Ulysses S. Grant
Rutherford B. Hayes
James A. Garfield
Chester Arthur
Grover Cleveland
Benjamin Harrison
Grover Cleveland

 • • •

George Bush
Bill Clinton
George W. Bush
Barack Obama
```

Now you're ready to quicksort some names! Input text file **presidents.txt** shown on the left contains the names of all the U.S. presidents in the order that they served.

Note that Grover Cleveland served two nonconsecutive terms.

Class **`Presidents`** reads the presidents' names and prints them in their original order. It should use quicksort to sort the names by last name, and then it prints the names in sorted order. Grover Cleveland should be listed twice in the sorted list.

If two presidents have the same last name, sort them by their first names. If they also have the same first names, then sort them by their middle names. For example, George Bush should be sorted before George W. Bush.

You will complete classes **`Name`** and **`NameSorter`**, and read the names into an **`ArrayList<Name>`**.

You must use quicksort and the partitioning method from your draft to sort the array list. The textbook uses quicksort to sort an array of integers, so you will need to make some adjustments to sort an array list of names.

You should print trace output for testing, but your final submission must not have any trace output.

File **Presidents.java**: http://www.cs.sjsu.edu/~mak/CS46B/assignments/7/Presidents.java
File **presidents.txt**: http://www.cs.sjsu.edu/~mak/CS46B/assignments/7/presidents.txt
File **namesoutput.txt**: http://www.cs.sjsu.edu/~mak/CS46B/assignments/7/names.out

Final **President.java** Codecheck URL**:**
http://codecheck.it/codecheck/files/1507040517919iude2tuhn0jfio6q8k87xr

Canvas: **Homework 7 Final**
Due: Friday, July 10 at 11:59 PM