CS 284: Homework Assignment 6
Due: May 11, 11:59pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

Late Policy. Late submissions are allowed with a penalty of 2 points per hour past the deadline.

2 Assignment (11 pts)

For this assignment, you will implement Lamport's iterative (non-recursive) variant of Hoare's quicksort sorting algorithm in Java. Bertrand Meyer provides a description of the algorithm on his blog[1] and references a video[2] showing Leslie Lamport's original description.

The concept is to maintain a set of pairs of indices which represent the bounds of the subarrays yet to be sorted. Initially, the set only contains the indices which bound the array that is to be sorted. In each iteration, a pair of bounds is removed from the set and if the corresponding subarray contains more than one element, quicksort's partition algorithm is carried out on the elements within the subarray. Partitioning produces two subarrays whose bounds are added to the set. The algorithm terminates, when the set is empty. At this point, the whole array will be sorted.

The signature of the main class should be public class `Sort` and is supposed to contain the following method:

```
public static <T extends Comparable<T>> void sort(T[] array)
```

[1] `https://bertrandmeyer.com/2014/12/07/lampsort/`
[2] `http://channel9.msdn.com/Events/Build/2014/3-642`

The sort method sorts the given array in increasing order and in-place, i.e., it sorts the elements in the given array rather than storing the sorted array elements in a newly allocated array.

Create an inner class with the signature `private static class Interval` with the data fields, constructors, and methods described below.

- Data fields:

```
private int lower;
private int upper;
```

- Constructors:

```
public Interval(int lower, int upper)
```

- Methods:

```
public int getLower() // returns the lower bound
public int getUpper() // returns the upper bound
public boolean equals(Object o) // returns true if this interval and the
//given interval have the same lower and upper bounds
public int hashCode() // returns lower * lower + upper
```

Use the median-of-three method discussed in class (*cf.* slides 13-15 of `Sorting3.pdf`) as your algorithm to select the pivot element. For median-of-three method, sort the elements at positions first, middle, and last in increasing order, apply two iterations of bubble sort (resulting in three comparisons and three potential swaps overall). If the array has size 3, then you are done. Otherwise, after the sort, swap the median of the three elements (now at position middle) with the element at position first. Use the element at position first (now the median of the three elements) as your pivot element. If your subarray contains only two elements, compare the elements and swap them if necessary. As subarrays with one element are already sorted, there is no need to add the bounds to the set.

To carry out the partition follow the approach from the recitation utilizing an up index and a down index.

Instead of making recursive calls on the subarrays, add the bounds (interval) of the subarrays to the interval set.

Write a class with the signature public class `SortTest` which contains several `JUnit` tests for your sort method. Test your sort method using various input arrays.

2.1 Hints

The documentation of the Java Set interface is available from `https://docs.oracle.com/javase/8/docs/api/java/util/Set.html`. Use `HashSet` as the implementing class of `Set` (i.e., every `Set` object should be an instance of `HashSet`).

3 Extra Credit Option (2.5 pts)

Implement *Counting Sort*. This algorithm takes as input an array of n integers in the range of 0 to k. Its running time is $\mathcal{O}(n)$, and hence is more efficient than all algorithms seen in class. In contrast to those seen in class, *Counting Sort* does not use comparisons.

The signature of the main class should be public class `CountingSort` and is supposed to contain the following method:

```
public static void sort(int[] A)
```

Next we present the pseudocode for `COUNTING_SORT (A, B, k)`, taken from **Introduction to Algorithms, 3rd Edition** by Cormen et al. `A` is the array `A[1..n]` to be sorted and has length n. Also, its elements are integers in the range $0..k$. `B` is an array `B[1..n]` that will hold the sorted output. `C` is an auxiliary array `C[0..k]` for temporary storage. You should use this as the basis for your Java implementation. Once you have implemented it, include some test cases.

```
for i = 0 to k do
    C[i] = 0
for j = 1 to n do
    C[A[j]] = C[A[j]] + 1
//C[i] now contains the number of elements equal to i
for i = 2 to k do
    C[i] = C[i] + C[i-1]
// C[i] now contains the number of elements <= i
for j =  n downto 1 do
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

Here is an example of a run. The array to be sorted is A whose size is $n = 8$.

A	2	5	3	0	2	3	0	3

The integers in A are in the range 0 to $k = 5$. Here is a trace:

- At line 5:

C	2	0	2	3	0	1

- At line 8:

C	2	2	4	7	7	8

- Iteration 1 of loop in line 9:

B							3		C	2	2	4	6	7	8

- Iteration 2 of loop in line 9:

B		0					3		C	1	2	4	6	7	8

- Iteration 3 of loop in line 9:

B		0				3	3		C	1	2	4	5	7	8

- After all iterations of loop in line 9:

B	0	0	2	2	3	3	3	5

4 Submission instructions

Please archive your Sort.java and the SortTest.java files in a single ZIP file and sumbit your ZIP file on Canvas. No report is required. If you completed the extra credit option, include CountingSort.java and CountingSortTest.java in the same ZIP. Some further guidelines:

- Use JavaDoc to comment your code. Partial credit will be given for comments, readability and style.
- Check the arguments of methods.