Assignment 1

Tools & Techniques for Large-Scale Data Analytics (CT5105)

NUI Galway, Academic year 2016/2017, Semester 1

- ***Submission deadline (strict): Sunday, 25th September, 23:59***
- *Please put all your answers and code files into a single .zip archive with name "YourName_Assignment1.zip" and submit via Blackboard*
- *Include all source code files (that is, files with name ending .java) required to compile and run your code, including all classes, interfaces, etc.*
- *Unless specified otherwise in the question, use only plain Java for this assignment (no external libraries or frameworks)*
- *Please note that all submissions will be checked for plagiarism*
- ***Use comments to explain your source code. Missing or insufficient comments can lead to mark deductions***

Remark: Some parts of this assignment require knowledge from next week's lecture, but you can already work on Q1 and Q2 (and perhaps Q3).

You are given the following implementation of a *distributed* sorting algorithm (Bucket Sort):

```
public static int[] bucketSort(int[] numbers, int bucketCount) {

    if (numbers.length <= 1) return numbers;

    int maxVal = numbers[0];
    int minVal = numbers[0];

    for (int i = 1; i < numbers.length; i++) {
        if (numbers[i] > maxVal) maxVal = numbers[i];
        if (numbers[i] < minVal) minVal = numbers[i];
    }

    double interval = ((double)(maxVal - minVal + 1)) / bucketCount; // range of bucket

    ArrayList<Integer> buckets[] = new ArrayList[bucketCount];

    for (int i = 0; i < bucketCount; i++)  // initialize buckets (initially empty)
        buckets[i] = new ArrayList<Integer>();

    for (int i = 0; i < numbers.length; i++)  // distribute numbers to buckets
        buckets[(int)((numbers[i] - minVal)/interval)].add(numbers[i]);

    int k = 0;

    for (int i = 0; i < buckets.length; i++) {

        Collections.sort(buckets[i]);  // calls Java's built-in merge sort (as a kind of "helper" sort)

        for (int j = 0; j < buckets[i].size(); j++) { // update array with the bucket content
            numbers[k] = buckets[i].get(j);
            k++;
        }
    }

    return numbers;
}
```

PTO

Q1.
Put the given method into a class and add a `main`-method which calls method `bucketSort` with some array of numbers and prints both the original array and the resulting sorted array of numbers. Run the given code and find out how it works (the Eclipse debugger might help) and what the basic idea behind this algorithm is.

[5 marks – for Q1 you only need to submit your class with the given code and your `main`-method]

Q2.
Modify the given code so that it makes use of *parallel* computing using multithreading, where appropriate. Use "traditional" Java style for this (that is, code which also works with Java version 6 or 7). Also provide a `main`-method which tests your modified `bucketSort` method with a few arrays of numbers and prints the original and the sorted arrays.
(Remark: No need to install Java 6 or 7 if you have already Java 8 installed – you can change the code style in Eclipse using setting Project -> Java Compiler -> Enable project specific settings -> Compiler compliance level.)

[40 marks]

Q3.
The given code uses a "helper" sorting method. More precisely, it calls *merge sort* from the Java API (the `Collections.sort(buckets[i])` call). However, in principle, any "helper" sorting method could be used here, not only merge sort.
Modify your code from Q2 (or the given code in case you couldn't do Q2) so that method `bucketSort` has an additional parameter `helperSortFunction` which represents any "helper" sorting function. In the body of your modified `bucketSort`, the sorting function represented by parameter `helperSortFunction` should be called instead of `Collections.sort`. This way, your modified `bucketSort` method should be able to use *any* "helper" sorting function specified by the caller of `bucketSort`.

An example call of `bucketSort` could then be `bucketSort(someArrayOfNumbers, 10, myHelperSortFn)`. However, your modified `bucketSort` should work with any helper sorting function passed into it as third argument, not only with the function represented by `myHelperSortFn`.

Again, use only Java 6/7-style code for this question. Hint: Anonymous classes and function objects...

Also, provide again a `main`-method which tests your modified `bucketSort` code with a few arrays of numbers and prints the original as well as the sorted arrays.

Remark: it is not required that you implement some new "helper" sorting function. In your `main`-method, you can simply test your code by passing `Collections.sort` as "helper" sorting function (in a suitable form) into the modified `bucketSort` (but remember that your code needs to work with any other "helper" sorting functions too).

[25 marks]

Q4.
Modify your solution for Q3 so that a Java 8 Lambda expression is used instead of an object of an anonymous class. That is, argument `helperSortFunction` of method `bucketSort` should now be a Lambda expression.
Also, provide again a `main`-method where you test your code by calling `bucketSort` with some arrays of numbers.

[30 marks]