

## CS 253 (Spring 2024)

### Assignment 3: Advanced Sorting

#### Goals

- Implement counting sort, radix sort and shellsort.
- Compare the average running times of various sorting algorithms on inputs of different sizes.

#### Implementation of Counting Sort:

The starter code contains a method called `countingSort(long[] a)`. You must fill in this method with code that implements a counting sort to sort an array of long integers. The input array  $A$  will contain nonnegative integers between 0 and  $k$  for some value of  $k > 0$ . Additionally, the algorithm requires two additional arrays:  $B$  of the same length as  $A$  to copy the elements to in sorted order and  $C$  of length  $k$  to store the frequencies.

#### Pseudocode for Counting Sort Algorithm

Input: Array  $A$  containing integers between 0 and  $k > 0$  for some value  $k$ .

Output: The array  $A$  will be sorted.

```
for  $j = 1$  to  $A.length$  do
     $C[A[j]] \leftarrow C[A[j]] + 1$ 
end for
for  $i = 1$  to  $k$  do
     $C[i] \leftarrow C[i] + C[i - 1]$ 
end for
for  $j = A.length$  to 1 do
     $B[C[A[j]]] \leftarrow A[j]$ 
     $C[A[j]] \leftarrow C[A[j]] - 1$ 
end for
```

#### Radix Sort

The starter code contains a method called `radixSort(long[] a)`. You must fill in this method with code that implements an LSD radix sort to sort an array of long integers. This sort should use counting sort on each digit of the numbers, starting with the least significant.

#### Shellsort

The starter code contains a method called `shellSort(long[] a)`. You must fill in this method with code that implements shellsort to sort an array of long integers. You should create a helper method to generate the sequence of gaps for the algorithm.

#### Pseudocode for ShellSort

```
for all  $h$  in a decreasing sequence of gaps do
    for  $i = h$  to  $n - 1$  do
         $key \leftarrow A[i]$ 
         $j \leftarrow i$ 
        while  $j \geq h$  and  $A[j - h] > key$  do
             $A[j] \leftarrow A[j - h]$ 
             $j \leftarrow j - h$ 
        end while
         $A[j] \leftarrow key$ 
    end for
end for
```

## Comparing Sorting Algorithms:

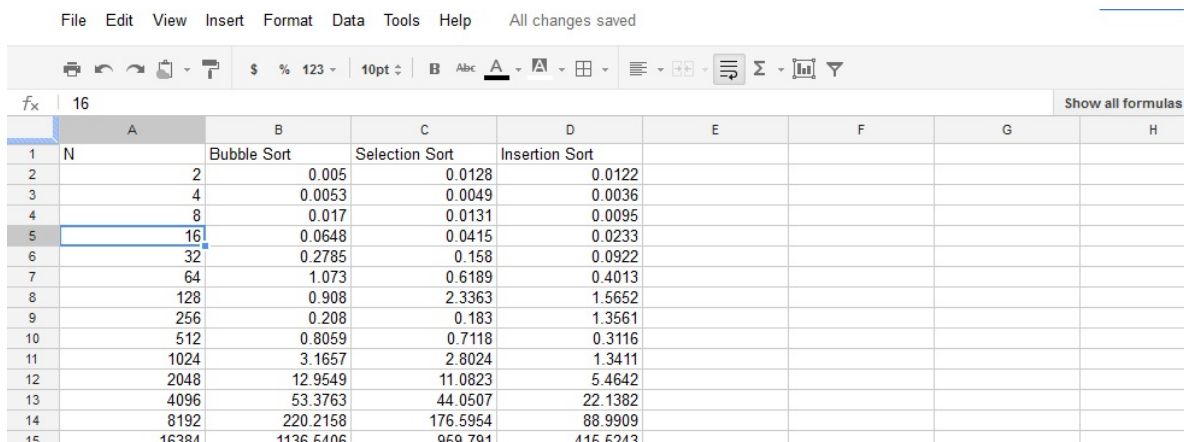
You will compare the average performance of six different sorting algorithms: selection sort, insertion sort, mergesort, counting sort, radix sort and shellsort. The selection sort, insertion sort and mergesort are provided in the starter code. The code for running the comparisons for different input sizes are provided in the main method. For each algorithm, it tests different input sizes (default is 16 which tests the first 16 powers of 2), and the input data are randomly generated. It performs each test multiple times (default is 5) in order to get an average running time. Afterward, it will print the results as a table: each row reports the input size (N), and the average running time for each algorithm in milliseconds. Familiarize yourself with the code in the main method and make sure you understand how the experiments are carried out. Run the main method and examine the table to compare the run times for the various algorithms. You can increase the value of the max variable if it helps with your analysis.

Note that the starter code checks after each sorting algorithm whether the sorting result is correct. If your sorting implementation produces incorrect sorting results, a message 'The sorting is INCORRECT!' will be printed out. So watch for such messages. Points will be deducted if your sorting result is incorrect.

## Plotting the running times:

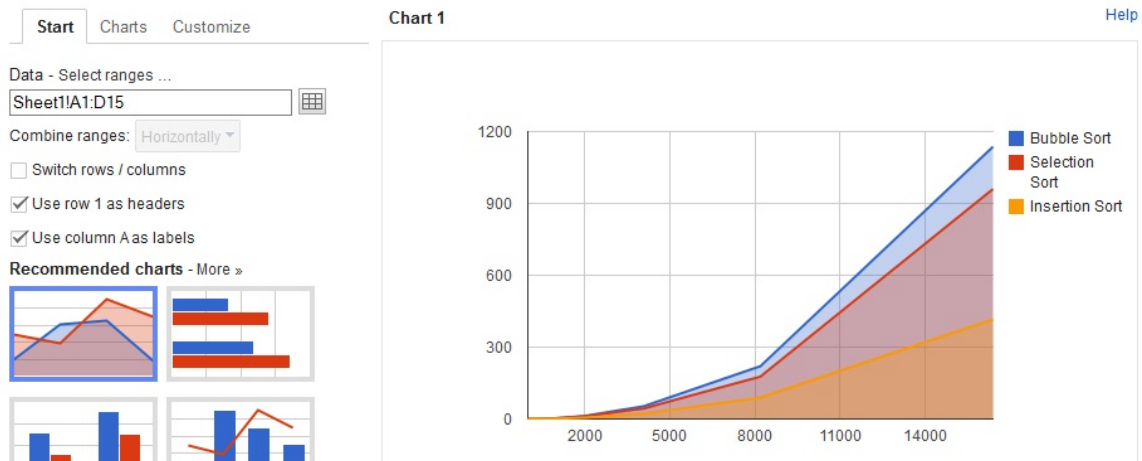
After the program prints out the table, plot the results in graphs. You can use Google SpreadSheet, Microsoft Excel, OpenOffice, or other tools. The following are the steps using Google Spreadsheet (which plots the results for 3 algorithms). If you use Microsoft Excel, or OpenOffice Calc, the steps should be similar. Just make sure that you select "Line chart" as your chart type and save your chart as an image (any image format is fine: .jpg, .png, .bmp, etc.) with a name Sorting.

- (a) Copy the running times reported to a spreadsheet.



	A	B	C	D	E	F	G	H
1	N	Bubble Sort	Selection Sort	Insertion Sort				
2	2	0.005	0.0128	0.0122				
3	4	0.0053	0.0049	0.0036				
4	8	0.017	0.0131	0.0095				
5	16	0.0648	0.0415	0.0233				
6	32	0.2785	0.158	0.0922				
7	64	1.073	0.6189	0.4013				
8	128	0.908	2.3363	1.5652				
9	256	0.208	0.183	1.3561				
10	512	0.8059	0.7118	0.3116				
11	1024	3.1657	2.8024	1.3411				
12	2048	12.9549	11.0823	5.4642				
13	4096	53.3763	44.0507	22.1382				
14	8192	220.2158	176.5954	88.9909				
15	16384	1136.5406	959.791	415.5243				

- (b) Click on "Insert" -> "Chart", and select "Use row 1 as headers" and "Use column A as labels." Select "Line" chart, and optionally, click on "Customize" and select "Medium" for Point Style. Feel free to add labels (such as chart title), to make the chart more informative. Examine the chart. This chart will help you visualize the difference of the running times of all the sorting algorithms, and also the trend when N grows large. Then click on "Insert" to insert the chart. Click on the drop down menu from the chart and select "Save image" to save the image to a disk file, and name it Sorting.png. In the future, when you perform scientific experiments, you will frequently plot your data as charts. So take your time to experiment with different types of charts and different ways of plotting them.



## Getting Started

1. Get the starter code `Sorting.java` from Canvas.
2. Complete the methods `countingSort()`, `radixSort()` and `shellSort()`.
3. Run the starter code to compare the running times for different sorting algorithms (you can change the parameters “max” and “runs” for testing purposes).
4. Plot the performance charts. You should create enough to visually demonstrate the comparisons between the running time of all algorithms (but not too many, 4 should be enough).

## Honor Code

The assignment is governed by the College Honor Code and Departmental Policy. Please remember to have the following comment included at the top of the file.

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING ANY
SOURCES OUTSIDE OF THOSE APPROVED BY THE INSTRUCTOR. _Your_Name_Here_
*/
```

## Submission:

Submit your completed `Sorting.java` file, as well as the tables and plots to Gradescope.

## Grading:

- Correctness and robustness: your sorting methods works correctly for any input array. (60 pts.)
- Efficiency: your sorting methods are efficient, and the running time of counting and radix sort is less than the mergesort and the running time of shellsort is less than insertion sort and selection sort, in most cases when  $N > 512$  (15 pts.)
- Code clarity and style (5 pts)
- Correctness of the plots: you included a running time chart and the data is correctly plotted. (20 pts.)