University of Wollongong
School of Computer Science and Software Engineering

CSCI124/MCS9124 Applied Programming Spring 2015

Lab 4 (1 mark)
Due 11:59pm Thurs 2nd April

Background:
In this lab you will analyse a sorting algorithm.

Task:
In the lectures we discussed how a programmer can perform their own analysis of a sorting algorithm by counting the number of comparisons and plotting them against sample size. For this lab exercise we will implement a sorting algorithm, counting the number of comparisons, and compare its performance to bubble sort.

The Algorithm:
This algorithm is in some way similar to bubble sort in that it compares pairs of elements of the array of data to be sorted and exchanges them if they are the wrong way around. However, this algorithm doesn't compare adjacent values – well not all the time. Each pass though the array, it performs bubble-like comparisons between values a fixed distance apart. It repeats this process for that distance until no swaps occur. The next pass then halves the distance between pairs, then halves again, until finally comparing the values one apart. Here is an illustration. Consider the data

```
13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10
```

Let's start with a distance of 5. We are then going to compare the first with the 6th, 2nd with the 7th, 3rd with the 8th,… up until 11th with 16th. Repeat until no swaps. This turns out to be equivalent to rearranging the data into 5 columns.

```
13 14 94 33 82
25 59 94 65 23
45 27 73 25 39
10
```

Note that the first column now contains the 1st, 6th, 11th and 16th. These four values make a group that is in fact bubble sorted. So sort each column.

```
10 14 73 25 23
13 27 94 33 39
25 59 94 65 82
45
```

With conventional Bubble Sort, it would have taken a lot of comparisons to get 10 to the top, but this needs less as we are only sorting 4 values. Now we halve the number of columns to 2.

```
10 14
73 25
23 13
27 94
33 39
25 59
94 65
82 45
```

We sort the two columns, then make a final one column to sort using Bubble Sort.
This sounds like a lot of work relative to Bubble Sort which is a poor sort algorithm, let's see if it is bad.
Here's the algorithm for sorting an array X of length N.

```
Determine an initial distance D between comparisons.
WHILE the distance D is greater than 0
        REPEAT
                FOR J=0 TO N-1-D
                        if X[J] > X[J+D]
                                swap X[J] and X[J+D]
        UNTIL no swaps occur in the for loop
        halve D
```

The initial value of D does impact how well this works. We'll use the starting value is the largest value of the form 2^m-1 that is still less than N. For example, if N = 100, then the starting value would be 63, since 64 - 1< 100 <= 128 - 1.

Procedure:

In the file `sort.cpp`, implement the function with prototype

```
void Sort(int array[], int n, int& ncomp);
```

which uses the above sort to sort the data in `array[]` of length `n`, returning the number of comparisons in the third argument. To see how the comparisons are counted, the file `bubblesort.cpp` on the website contains the implementation of Bubble Sort to perform the sort.

To test the sort algorithm, the file `L4main.cpp` contains a driver program to generate random shuffles of 100, 200, 300,…,10000 integers and call the sort function to sort them. It generates a data file ready for a graph converter called `converter2` (you can compile converter2.cpp similarly to converter in Lab 3) which makes it ready for plotting.

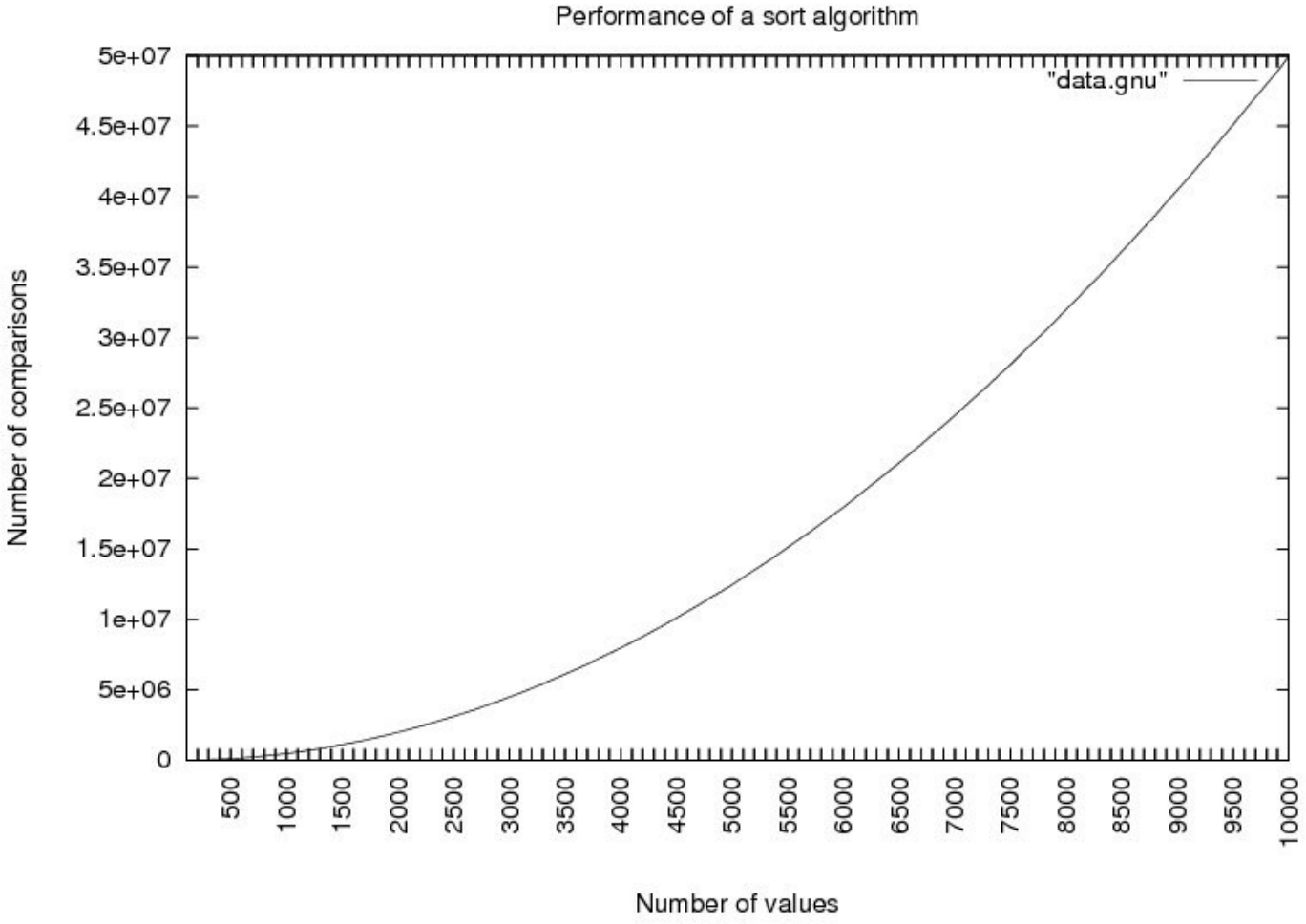
```
$ g++ L4main.cpp sort.cpp -o sort
$ ./sort > data.txt
$ ./converter2 data.txt data.gnu | gnuplot
```

The driver program can also be used to check if your sort is working by printing out the sorted data – remember there's a lot of data so use `less` to control it. To compile this version of the program define PRINT by compiling with

```
$ g++ -DPRINT L4main.cpp sort.cpp
```

If you use `bubblesort.cpp` instead of your own in the above compile, then run it, then the use of `converter2` will produce the following graph or something like it.

Note the curve of the graph. Remember that Bubble Sort is $O(n^2)$, so we expect a graph like this.



Now try it with your own implementation of this "improved" sort and see what the result is. You will want to rename the output file first. Can you guess at its performance? Is it better or worse than bubble sort?

Submit:

You are to submit `sort.cpp` using

```
$ submit –c csci124 –a lab4 –u <username> sort.cpp
```