Joanna Klukowska
joannakl@cs.nyu.edu

Project 5: Sort Algorithms Evaluation

Due date: April 30, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will evaluate performance of different sort algorithms. The source code (based on the code in the textbook) is given to you and you only need to make slight modification to the code. Your main task is to collect data and prepare a written report about your findings.

Objectives

The goal of this project is for you to master (or at least get practice on) the following tasks:

- reading existing code
- understanding and evaluation of sorting algorithms
- comparison of performance of different sorting algorithms

Start early! This project may not seem like much work, but collection of the data needed and preparation of the report might take some time, so do not leave it for the last moment.

Source Code

The textbook provides a class that implements several different sorting algorithms:

- two different bubble sorts
- selection sort
- insertion sort
- merge sort
- quick sort
- heap sort

You should study the code for each of them so that you understand how each works. You will need to make slight changes to all of the implementations to collect data about their performance. In addition, you will need to make changes to the `main` function to display the results collected by the sorts. There are ways to changing the `main` function to provide full (or almost) full automation of the data collection. You are not required to so so and you can brute-force your way through making small changes in the code and re-running the program to collect each required result. It is completely up to you how you collect the data - your grade will not be affected by changes that you make to the `main` function (it will be affected by the changes that you make to the source code of the sort methods).

Performance Evaluation

Some sorting algorithms are more efficient than others because they perform fewer comparisons and data swaps in order to achieve their goal. In this project you will be evaluating the number of comparisons involving data elements in the array to be sorted. To do that, you

Joanna Klukowska
joannakl@cs.nyu.edu

need to add code to each of the sort implementations so that the number of actual comparisons is counted when an array is sorted. (Note that since you are not looking at the number of swaps, but only at the number of comparisons, some of the results may seem surprising.)

Your analysis for each algorithm should report on the number of comparisons needed to sort arrays of increasing sizes. Those values should follow expected patterns suggested by the asymptotic analysis of each of the sorting algorithms.

For all sorts, your report should contain the data (i.e., the number of comparisons required) for the sizes of arrays ranging from 10,000 elements to 100,000 elements in increments of 10,000.

For the faster sorts, $O(N \log N)$, you should also include the data for sizes of arrays ranging from 100,000 elements to 1,000,000 elements in increments of 100,000. (Feel free to try to run the quadratic sorts on the arrays of those sizes, but it may take a long while before you see the results, so you are not obligated to report on those).

Extra Credit (20 points)

Provide a similar analysis based on the number of swaps performed by each algorithm. The tables and charts generated based on the number of swaps should and their discussion should be incorporated into relevant parts of the report.

Report

Your report should contain 1) discussion of each algorithm individually, 2) comparison of all the $O(N^2)$ sorts, 3) comparison of all the $O(N \log N)$ sorts. For the individual discussion of each algorithm you should provide the data in form of a table and in form of a chart. The chart should include the plot of N^2 or $N \log N$, whichever one is appropriate. You should also include a brief paragraph of discussion and observations.

The comparison of the algorithms that are in the same class from the point of view of asymptotic analysis (either $O(N^2)$ or $O(N \log N)$) should contain a chart showing data for all three algorithms within that class as well as the plot of of N^2 or $N \log N$. This chart should be accompanied by discussion of relative performance of the relevant algorithms (performance from the point of view of the number of comparisons performed).

All charts and tables should be clearly labeled.

Report Format

The report needs to contain the following sections

title page should contain your name, netID, course number and section number; it should not contain any charts or descriptions

individual descriptions - the descriptions and data of six individual algorithms; make sure that those appear one per page; (if you wish to write more than one page for any algorithm this is fine, but make sure to start each section on a new page)

$O(N^2)$ **sorts summary**

$O(N \log N)$ **sorts summary**

The report has to be in a format of a written document and it has to be typed. We will not accept scanned hand-written reports. We will not accept reports written in a spreadsheet format (you should use a spreadsheet to generate your charts and to create tables, but that information has to be incorporated into a regular document).

Example Report

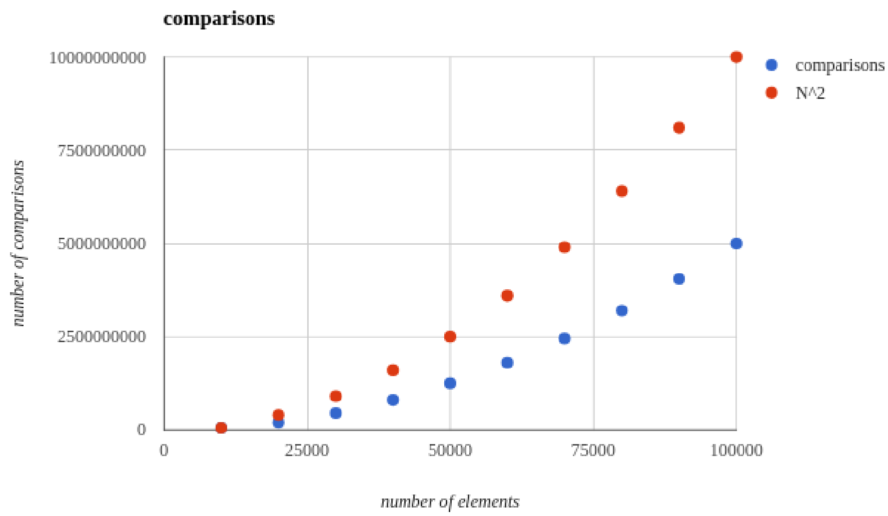
The following section shows an example of tables, charts and discussion that should be included for each algorithm. Your report should contain similar discussion for the other bubble sort implementation and the other five algorithms.

Note: this includes the discussion of extra credit data generated for number of swaps in the array.

Joanna Klukowska
joannakl@cs.nyu.edu

Bubble Sort

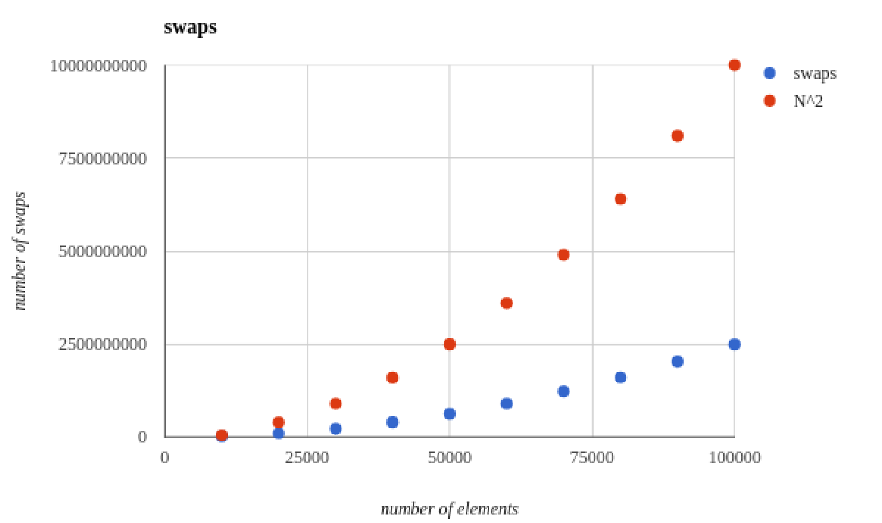
N	comparisons	N^2
10000	49995000	100000000
20000	199990000	400000000
30000	449985000	900000000
40000	799980000	1600000000
50000	1249975000	2500000000
60000	1799970000	3600000000
70000	2449965000	4900000000
80000	3199960000	6400000000
90000	4049955000	8100000000
100000	4999950000	10000000000



The number of comparisons performed by the bubble sort algorithm implementation follows its theoretical performance of $O(N^2)$. The number of comparisons performed is exactly the same regardless of the data in the array. In fact, if the array is sorted prior to call to `bubbleSort()` function, the number of comparisons performed by `bubbleSort()` remains the same. It is always equal to $1/2N(N-1)$.

The number of swaps performed varies depending on the content of the array. The table and chart below show the results for arrays with randomly arranged elements. When the `bubbleSort()` function is given an array that was sorted, the number of data swaps is zero. When the `bubbleSort()` function is given an array that was reverse-sorted, the number of data swaps is largest and it is equal exactly to $1/2N(N-1)$ (same number as the number of comparisons).

N	swaps	N^2
10000	25333672	100000000
20000	99394878	400000000
30000	225709001	900000000
40000	402256534	1600000000
50000	624944393	2500000000
60000	900889805	3600000000
70000	1226638222	4900000000
80000	1603017937	6400000000
90000	2026992496	8100000000
100000	2498027476	10000000000



Joanna Klukowska
joannakl@cs.nyu.edu

Grading

10 points times 6 section for each of the six algorithms (this includes the tables, charts, discussion and the change in source code that was made to count the number of comparisons)

20 points summary section for the $O(N^2)$ sorts

20 points summary section for the $O(N \log N)$ sorts

20 points optional extra credit

How and What to Submit

Your should submit your modified source code file to NYU Classes - it should be a single uncompressed file.

You should submit the report to Gradescope!