CAB301 Algorithms and Complexity
Assignment 2 — Empirical Comparison of Two Algorithms

Due: Sunday, 28th May 2017

Weight: 40%

Group or individual: Group of two students

Summary

In the first assignment you learned how to analyse an algorithm experimentally, by implementing it as an executable program, measuring the program's run-time characteristics, and comparing the results with theoretical predictions for the algorithm. In this assignment you will undertake a similar exercise, but this time the aim is to directly compare two different algorithms that both perform the same function but may have different efficiencies. In particular, you must ensure that both programs are analysed under exactly the same conditions, to ensure that the results are meaningfully comparable. As in the first assignment, you will be required to count the number of basic operations performed, measure execution times, and produce a clear report describing your findings. (An example report is available on the CAB301 BlackBoard site.)

Tasks

To complete this assignment you must submit a written report, and accompanying evidence, describing the results of your experiments to compare the efficiency of two given algorithms. The steps you must perform, and the corresponding (brief) summaries required in your written report, are as follows.

1. You must ensure you understand the algorithms to be analysed.
   - Your report must briefly describe the two algorithms.
2. You must define a common basis for meaningful comparison of the two algorithms, in terms of basic operations and the size of inputs.
   - Your report must describe your choice of 'basic operation' applicable to both algorithms.
   - Your report must describe your choice of 'problem size' applicable to both algorithms.
   - Your report must summarise the predicted (theoretical) time efficiency of the two algorithms, as far as possible. For the purposes of this assignment we are interested in average-case efficiency only.
3. You must describe the methodology you used for performing the experiments.
   - Your report must describe your choice of computing environment.
   - Your report must describe your C++ implementation of the given algorithms. You must ensure that the correspondence between features of the algorithms and the corresponding program code is clear, to confirm the validity of the experiments. The program code should replicate the structure of the algorithms as faithfully as possible.
   - Your report must explain how you showed that your programs work correctly. Thorough testing would normally be sufficient.
   - Your report must explain clearly how you produced test data for the experiments, or chose cases to test, as appropriate. Usually this will involve generating random data for different sizes of input. In particular, it is important that both algorithms are tested using the same data, to ensure that the comparative results are meaningful.
   - Your report must explain clearly how you counted basic operations, e.g., by showing the relevant program code for maintaining 'counter' variables. In particular, it should be easy to see that the method used is accurate with respect to the original algorithms.

- Your report must explain clearly how you measured execution times, e.g., by showing the relevant program code augmented with calls to 'clock' or 'time' procedures. Since it is often the case that small program fragments execute too quickly to measure accurately, you may need to time a large number of identical tests and divide the total time by the number of tests to get useful results.

4. You must describe the results of your experiments.

   - Your report must *briefly* explain how you proved that your programs work correctly. This would normally be done through testing.

   - Your report must show in detail the results of comparing the number of basic operations performed by the two algorithms for different sizes of inputs. The results should be plotted as one or more graphs which make it easy to see how the two algorithms compare. You must produce enough data points to show a clear 'trend' in the outcomes. It must be clear how many individual data points contributed to the lines shown and how many individual tests contributed to each data point. You must briefly explain what the results reveal about the comparative efficiency of the two algorithms.

   - Your report must show in detail the results of comparing the execution times of the two programs for different sizes of inputs. The results should be plotted as one or more graphs which make it easy to see how the two algorithms compare. You must produce enough data points to show a clear 'trend' in the outcomes. It must be clear how many individual data points contributed to the lines shown and how many individual tests contributed to each data point. You must briefly explain what the results reveal about the comparative efficiency of the two programming language implementations of the algorithms.

5. You must produce a brief written report describing all of the above steps.

   - Your report should be prepared to a professional standard and must not include errors in spelling, grammar or typography.

   - You are free to consult any legitimate reference materials such as textbooks, web pages, etc, that can help you complete the assignment. However, you must appropriately acknowledge all such materials used either via citations to a list of references, or using footnotes. (Copying your assignment from one of your classmates is *not* a legitimate process to follow, however. Note the comments below concerning plagiarism.)

   - Your report must be organised so that it is easy to read and understand. The main body of the report should summarise what you did and what results you achieved as clearly and succinctly as possible. Supporting evidence, such as program listings or execution traces, should be relegated to appendices so that they do not interrupt the 'flow' of your overall argument.

   - There should be enough information in your report to allow another competent programmer to fully understand your results. This does not mean that you should include voluminous listings of programs, execution traces, etc. However, you should include the important parts of the code, and brief, precise descriptions of your experimental methodology.

6. You must provide all of the essential information needed for someone else to duplicate your experiments in electronic form, including complete copies of program code (because the written report will normally contain code extracts only). As a minimum this data must include:

   - An electronic copy of your report;

   - The complete source code for your C++ implementations of the algorithms; and

   - The complete source code for the test procedures used in your experiments.

   Optionally you may also include electronic versions of the data produced by the experiments. In all cases, choose descriptive folder and file names.

Algorithms

Levitin presents the following algorithm for finding the distance between the two closest elements in an array of numbers [Ex. 1.2(9), p. 18].

Algorithm *MinDistance*$(A[0..n-1])$
//Input: Array $A[0..n-1]$ of numbers
//Output: Minimum distance between two of its elements
$dmin \leftarrow \infty$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    if $i \neq j$ **and** $|A[i] - A[j]| < dmin$
      $dmin \leftarrow |A[i] - A[j]|$
return $dmin$

He then asks if there is a more efficient algorithm to do this task. One possible answer is as follows.

Algorithm *MinDistance2*$(A[0..n-1])$
//Input: An array $A[0..n-1]$ of numbers
//Output: The minimum distance d between two of its elements
$dmin \leftarrow \infty$
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    $temp \leftarrow |A[i] - A[j]|$
    if $temp < dmin$
      $dmin \leftarrow temp$
return $dmin$

This version would appear to be more efficient because it compares the same pair of numbers only once, and avoids recomputing the same expression in the innermost loop. Can your empirical analysis confirm the claim that *MinDistance2* is more efficient than *MinDistance* on average?

Since the optimisation in *MinDistance2* is motivated by a desire to reduce the number of expressions that access elements of array *A*, you may want to use this principle to justify your choice of basic operation. (When counting basic operations for algorithm *MinDistance*, be careful to consider whether or not the second conjunct is evaluated when index *i* equals index *j*.)

Unfortunately, Levitin does not give an analysis of average-case efficiency for either algorithm. Nevertheless, it is obvious from inspection that the best- and worst-case behaviours for both algorithms are quadratic with respect to the length of the array. (Why?) Thus their average-case efficiencies must be quadratic as well. (Why?) However, although both algorithms belong to the same efficiency class, your experiments should reveal that algorithm *MinDistance*'s average-case efficiency function has a significantly larger 'constant multiplier' than that of *MinDistance2*.

Assessment Criteria

The specific criteria by which your assignment will be assessed are detailed in the Marking Schema and Feedback Sheet for this assignment. Assessment will be based primarily on your written report. However, if there is some concern about the originality of your work or the quality of your experimental results you may be asked to give a practical demonstration of your program.

Submission

Your assignment solution should be submitted electronically via Blackboard before 11:59 pm on 28th May 2017. Your submission should be a zip file. The file name should contain your student number and include the following items:

- All source code of your algorithm implementations
- A detailed report on your empirical analysis of both algorithms - the structure of your detailed report must be the same with that of the sample assignment
- Statement of completeness including the percentage contribution of each student