CAB301 Algorithms and Complexity

Assignment 1 — Empirical Analysis of an Algorithm

Due: Friday, 12th April 2019

Weight: 30%

Group or individual: Individual

Summary

Firstly you will identify the basic operation of a given algorithm and count the number of times the basic operation is performed by the algorithm, to confirm its predicted order of growth by experiments. Then you will measure its actual execution time, to determine whether the implementation introduces additional overheads (or optimisations!) not allowed for in the theoretical analysis. Finally, you must produce a detailed report describing your findings. This must all be done with a high degree of professionalism, as would be required when analysing an algorithm to be used in some critical application.

Tasks

To complete this assignment you must submit a written report, your implementation of a given algorithm in C#, C, C++, or Java, and a file containing all your original experimental results to measure a given algorithm's average-case efficiency. 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 algorithm to be analysed.
   - Your report must briefly describe the algorithm.
2. You must explain clearly the choice of basic operation and problem size for the particular algorithm of interest.
3. You must decide on an appropriate methodology, tools and techniques for performing the experiments.
   - Your report must describe your choice of computing environment. You must also say how you produced test data for the experiments, or chose cases to test, as appropriate.
4. You must implement the given algorithm in C#, C, C++ or Java, and verify its functional correctness.
   - Your report must describe your programming language implementation of the given algorithm. You must ensure that the correspondence between features of the algorithm and the program code is clear, to confirm the validity of the experiments. (For instance, implementing a recursive algorithm iteratively would not be acceptable because the time efficiency of the program may be very different from that of the algorithm. Similarly, certain code refactoring or optimisation may influence the experiments in undesirable ways.) The program code should replicate the structure of the algorithm as faithfully as possible.
   - Your report must explain how you showed that your program works correctly. (Thorough testing would normally be sufficient, although you may prefer to give a formal proof of correctness.)
5. You must count the number of basic operations performed by the program on a range of input values, and present the results in a form that can be compared easily against the theoretical

efficiency predictions. To do this you will need to: devise a way of counting basic operations, typically by incrementing a counter variable at the relevant point(s) in the code; run several tests, using appropriately chosen test data; plot the test outcomes on a graph; and state briefly how your experimental results compare to the predictions.

- Your report must explain clearly how you counted basic operations, e.g., by highlighting the relevant statements inserted into the program. In particular, it should be easy to see that the method used is accurate with respect to the original algorithm.
- You must perform enough experiments to produce a clear 'trend' in the outcomes. Your report must explain how you produced test data. Depending on the kind of algorithm involved, you may need to produce sets of 'random' values (so that you can produce average-case results for a particular size of input), or an ordered sequence of test values (so that you can show how the algorithm grows with respect to the input's size). In either case you may choose to create test data manually (which may be very tedious) or automatically (which may require some programming).
- You must present your experimental results as a graph. NB: You must state clearly how many data points contribute to the line(s) on the graph and what each data point represents. If possible, you should use a graph drawing tool that displays each data point as a distinct symbol.
- You must state whether or not the experimental results matched the predicted number of operations. If they do *not* match then you must offer some explanation for the discrepancy. (Normally we would expect that counting basic operations produces results that closely match the theoretical predictions, but it is possible that there is some peculiarity of your experimental set-up that skews the results, or even that the theoretical predictions are wrong.)

6. You must measure the execution time for your program on a range of input values, and present the results in a form that can be compared easily against the theoretical efficiency predictions. To do this you will need to: devise an accurate way of measuring execution times, typically by recording the system 'clock' time at relevant points in the code; run several tests, using appropriately chosen test data; plot the test outcomes on a graph; and state briefly how your experimental results compare to the predictions.
   - Your report must explain clearly how you measured execution times, e.g., by showing the relevant test program. (Alternatively, you may even choose to time your program with a stopwatch, although this is unlikely to produce accurate results.) It is often the case that small program fragments execute too quickly to time accurately. Therefore, 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.
   - You must perform sufficient experiments to produce a clear 'trend' in the outcomes. Your report must make clear how you produced test data (as per the discussion above on counting basic operations).
   - You must present your experimental results as a graph. NB: You must state clearly how many data points contribute to the results on the graph and what each data point represents. If possible, you should use a graph drawing tool that displays each data point as a distinct symbol.
   - You must state whether or not the experimental results matched the predicted order of growth. It is possible that your measured execution times may not match the prediction due to factors other than the algorithm's behaviour, and you should point this out if this is the case in your experiments. For instance, an algorithm with an anticipated linear growth may produce a slightly convex scatterplot due to operating system and memory management overheads on your computer that are not allowed for in the theoretical analysis. (However, a concave or totally random scatterplot is more likely to be due to errors in your experimental methodology in this case!)

7. You must produce a 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 another student 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 experimental methodology. This does not mean that you should include voluminous listings of programs, execution traces, etc. Instead you should include just the important parts of the code, and brief, precise descriptions of your methodology.
8. You must provide all of the essential information needed for someone else to duplicate your experiments in your submission, including complete copies of program code (because the written report will normally contain code extracts only), in addition to an electronic copy of your report in Microsoft Word or PDF.

Algorithm to be Analysed

ALGORITHM *BruteForceMedian*($A[0..n-1]$)
  // Returns the median value in a given array A of n numbers. This is
  // the kth element, where $k = \lceil n/2 \rceil$, if the array was sorted.
  $k \leftarrow \lceil n/2 \rceil$
  for i **in** 0 **to** $n-1$ **do**
    numsmaller $\leftarrow 0$ // How many elements are smaller than $A[i]$
    numequal $\leftarrow 0$ // How many elements are equal to $A[i]$
    for j **in** 0 **to** $n-1$ **do**
      if $A[j] < A[i]$ **then**
        numsmaller $\leftarrow$ *numsmaller* $+ 1$
      else
        if $A[j] = A[i]$ **then**
          numequal $\leftarrow$ *numequal* $+ 1$
    if *numsmaller* $< k$ **and** $k \leq$ (*numsmaller* + *numequal*) **then**
      return $A[i]$

The theoretical average efficiency of the above algorithm belongs to $\Theta(n^2)$.

Marking Criteria

The specific criteria by which your assignment will be assessed are detailed in the Marking Schema and Feedback Sheet for this assignment, which can be found in the Blackboard. 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 11:59pm on 12^{th} April 2019. Your submission should be a zip file. The file name should contain your student number and include the following items:

- An electronic copy of your report in Microsoft Word or PDF;
- Source code for all the programs used in your empirical analysis;