Fundamental Algorithms – A Guide to Laboratory Practice

General Requirements

For this laboratory you are asked to implement and analyze (practically) the correctness and efficiency of a number of algorithms. The programming language(s) are C/C++ the style is procedural. You are not allowed to use any additional libraries, containing existing implementations of necessary data structures or algorithms. The programming environment is Visual Studio 2008. You are encouraged to use the pseudo-code provided at the course/seminar to implement the algorithms, and employ any information from your course/seminar notes. Also, keep a copy of the *Introduction to Algorithms* book by Thomas Cormen close, since you will probably need to consult it often while trying to solve the assignments. For language-related information, use the *MSDN* library or the almighty *Google* search.

Assignments and Grading

Each assignment will be graded individually. At the end of the laboratory sessions (Week 14 probably), you will be given a quiz test consisting of a small number of questions from the assignments. The final mark at the laboratory is based on the average of the individual assignment grades. Starting from this average, a penalty can be applied for constant late delivery (check the delivery extensions sub-section). How well you do on the final quiz may influence the final laboratory grade by 1 point (+/-).

Assignment deliverables

- **Pseudo-code** on paper (your implementation should start from the pseudo-code)
- C/C++ procedural **implementation**, code should be commented
- The source file should have a header (block comments) containing:
    - Personal identification information (name, group)
    - Problem specification
    - Start and end date
    - Special evaluation requirements if necessary
    - **Conclusions and personal interpretations, running time, best/worst cases, memory, etc**
- **Running example(s)**: You must give a running example of your algorithm/procedures, on reasonably small-sized input(s)
- **Analysis**: generation and interpretation of output charts/tables, according to the individual assignment requirements

Assignment delivery

The majority of assignments are 1-week assignments. At the end of the session, you have to present your *code+analysis+interpretation* to the teaching assistant. If your assignment is not complete, you present what you have so far. You are allowed to continue working at your assignment and deliver it later (check the section on **delivery extensions**). When you consider that your assignment is complete, upload the C/C++ source file using the form provided on the website (*http://users.utcluj.ro/~cameliav/fa.php*, look for the icon corresponding to the assignment you want to upload).

If you have more than one source file for your algorithm (excluding the additional .h files you will be provided as supporting material during the lab sessions), upload a .zip archive containing all your source files to the same address. Do not include specific environment project files! The naming convention for the attachment is:

- *<NameSurname>-<Group>-<AssignNo>.c,*
- *<NameSurname>-<Group>-<AssignNo>.cpp*
- *<NameSurname>-<Group>-<AssignNo>.zip*

E.g.for attachment name: LemnaruCamelia-30222-1.zip

Assignment grading

Each assignment is graded incrementally – the requirements are to be completed incrementally. Each assignment has four grading thresholds: grades 5, 7, 9 and 10. The requirements for each grade are assignment specific, and you can consult them at http://users.utcluj.ro/~cameliav/fa/Assignment_thresholds.pdf

Delivery extensions

The majority of assignments are 1-week assignments. If, however, you do not finish by the end of the laboratory session, each assignment may have the following extensions:

- *Extension_1*: the assignment can be delivered at the beginning of the following laboratory session, at the beginning of the session
- *Extension_2*: For maximum grade 8, the assignment can be delivered at the beginning of the second following the original deadline (not all assignments have Extension_2)

Note: ! Constant delivery of assignments (>50%) at Extension_1 will be penalized with 1 point from the final mark.

!! Constant delivery of assignments (>50%) at Extension_2 will be penalized with 1 point at the final mark.

Missing laboratory sessions

All assignments have to be solved (to a certain extent). If, for some reason, you miss **one** laboratory session, you may solve the corresponding assignment at home and present it at the beginning of the following session. You must attend the laboratory with your group! (i.e. you are not allowed to attend a laboratory session with another teaching assistant).

If you miss more than 1 laboratory session (but not more than 3!), you will be asked to solve other assignments (and not the ones you have missed).

Coding Guidelines (the basics)

http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html
http://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html
http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Evaluation Tricks – how to evaluate the time complexity of your algorithms

For most assignments, you are asked to evaluate (perhaps comparatively), the running time of 1/several algorithms in the average case (and, for some assignments, in the best and worst case). You have to identify additional memory requirements also, and write such observations in the header of your source code!

However, before you pass on to the evaluation, make sure your algorithms work correctly!!!

Therefore, for each analysis case, you have to generate the according input data, of varying size (the superior limit in usually 10.000, the increment 100 – but this may differ for some assignments). For the average case, you have to repeat the measurement, at least 5 times, and report the average of your measurements. When you are asked to compare algorithms, make sure you test the algorithms in the same conditions! (same random input data for the average case, or the correct corresponding data for the best/worst cases).

What do you measure? When evaluating the running time, for each run of your algorithm, you have to count the number of operations performed by your algorithms, i.e. the number of assignments and the number of comparisons performed on the input structure, or on corresponding auxiliary variables (i.e. for sorting, no operations on indices or flags should be counted!!!). These measurements are saved in a file (usually .csv).

How do you analyze? Use the output data (.csv file) to generate required charts/tables, which will help you interpret the results. You may need, in some cases, to limit the *Ox* or *Oy* values on your charts, for better visualization of certain characteristics (overlapping, behavior on small sized inputs, asymptotic behavior, etc.)

Hint: feel free to be aggressively inquisitive with your charts, try to get as much out of them as you can (of course, do not try to see something that is not there). You will be rewarded accordingly for your efforts ☺

Write all your observations regarding the analysis in the analysis part of the Header section in your main source code file (provided you have more than 1 source files for your assignment).

Example: *Compare insertion sort and selection sort in the average case, for input sequences of sizes 100->10.000 (using an increment of 100)*.

Thus, you have to generate a random sequence for each intermediate size (100, 200, …, 10000). Then, apply both sorting methods (on the same sequence, independently) and count, for each, the number of assignments and the number of comparisons. Since we are dealing with the average case, repeat this 5 times, and write the average of the measurements in a .csv file, as you are shown in the *Introductory Laboratory session*. Then, using the results in the .csv file, generate three Excel scatter plots (you are shown how to do that in the *Introductory Laboratory session* as well): 1 comparing the number of comparisons used by the two methods, 1 for the number of assignments, and 1 for the overall number of operations performed by the two methods.

Don’t forget you also have to interpret the three charts (order of complexity, which method is better for small/large sized inputs, etc.) – and add these observations to the header of your source code file.