Faculty of
Technology, Engineering and the Environment

BIRMINGHAM City University

In Course Assessment Brief

Undergraduate Programme
Academic Year 2009/2010

Module:	**Data Structures and Algorithms UG2**
Assessment Title:	Sorting test suite advanced programming exercise.
Assessment Identifier:	**1.2**
School:	CTN
Module Co-ordinator:	Richard Kay
Assessment Details and Deadlines:	See ECMS My Course on the intranet.
Brief Assessment Details	Students will work individually on programs which implement advanced data structures and algorithms, and individually on reports which analyse the performance of implemented programs. The program comprises an automated test facility integrating various sort algorithms.
Learning Outcomes to be Assessed	Application of advanced algorithms and data structures, including recursion, binary trees, hash tables, randomisation, heaps.

If you should fail this module you will be permitted to be re-assessed on up to three occasions. If you fail to attend or to submit work for re-assessment at the next opportunity you will be deemed to have exhausted one of the opportunities.

IMPORTANT STATEMENT

Plagiarism: the presentation of the work of another (from whatever source: book, journal, internet etc) as if it were one's own independent work. This can be anywhere on a continuum ranging from sloppy paraphrasing to verbatim transcription without crediting sources.

You are advised to refer to the Student Handbook on matters of cheating and plagiarism as they relate to coursework, group assignments, class tests and examinations. Both cheating and plagiarism are totally unacceptable and the University maintains a strict policy against them. It is YOUR responsibility to be aware of this policy and to act accordingly.

The University requires that the following statement is included in all module documents.

> *"You are reminded of the University Disciplinary Procedures which refer to cheating. Except where the assessment of an assignment is group-based, the final piece of work which is submitted must be your own work. Close similarity between assignments is likely to lead to an investigation for cheating. It is not advisable to show your completed work to your colleagues or to share and exchange disks.*
>
> *You must also ensure that you acknowledge all sources you have used. Work which is discovered to be the result of collusion or plagiarism will be dealt with under the University's Disciplinary Procedures, and the penalty may involve the loss of academic credits.*
>
> *If you have any doubts about the extent to which you are allowed to collaborate with your colleagues, or the conventions for acknowledging the source you have used, you should first of all consult module documentation and, if still unclear, your module tutor."*

You will be asked to confirm in writing when handing in any piece of assessed work that it is your own by completing the Coursework Submission & Record Form which should be printed from ECMS My-course on https://mytid.bcu.ac.uk/.

It is the STUDENT'S responsibility to accurately complete the form and comply with its rules and guidance as described in the student handbook for this academic year.

Table of Assessment Criteria and Associated Grading Criteria

Assessment Criteria ◊	**1.** Sort key comparison and record move count instrumentation and test automation.	**2.** Program source code, quality and performance	**3.** Test plan, test data, test results as presented by user interface and within report.	**4.** Report analysing test results, development process and algorithm performance
Weighting:	20.00%	40.00%	20.00%	20.00%
Grading Criteria **0 – 29%**	Minimal or no effort and development. No understanding of algorithms evidenced.	Minimal or no effort and development. No knowledge evidenced.	Minimal or no effort and development. No knowledge evidenced.	Minimal or no effort and development. No knowledge evidenced.
30 – 39%	Little evidence of effort resulting in little or random instrumentation demonstrating little understanding.	Little evidence of effort resulting in little development. Nothing or very little useful working.	Little evidence of effort resulting in Insufficient development. No repeatability of results.	Little evidence of effort resulting in trivial or incoherent report and little knowledge.
40 – 49%	Some effort but large gaps and errors. Knowledge of algorithms limited and no attention to numeric overflow.	More than 1 sort implemented but poor overall quality and limited performance. At least one required sort algorithm missing..	Some test data available and results but large gaps and questionable test planning.	Enough understanding to pass but of limited quality. Large gaps demonstrating poor understanding and little analysis.
50 – 59%	most instrumentation requirements covered but with gaps and errors.	Program works in most respects but with limited performance. Coding questionable.	Limited range of planned and completed tests which cover most requirements .	Some analysis of algorithms and a fair understanding of performance issues demonstrated but some gaps.
60 – 69%	Good instrumentation applied to source and understanding of numeric overflow requirements some test automation but incomplete.	Working in nearly all respects with some but not consistently good performance. Good coding.	Good range of tests which are planned and results documented and presented but coverage not excellent.	Good knowledge and analysis of algorithms and performance issues demonstrated. Clear well-laid out report.
70+%	Fully accurate instrumentation, modular and with all numeric overflows trapped and intelligently handled. Full test automation.	Working in all respects with excellent performance. Excellent standard of coding.	Extensive tests planned and presented, and results documented demonstrating all required program facilities.	Full knowledge of algorithms and analysis and understanding of performance issues of an excellent standard. Very clear, complete and authentic report.

Assessment Details:

Students are required to design, implement test and document a program based on the program requirements stated below.

Deliverables

1. Your Sort key comparison and record move instrumentation

In all sort algorithms implemented your program will count the numbers of record moves and the number of key comparisons automatically. Counts will be kept accurately taking into consideration the need to avoid misreporting due to numeric overflows. It should be possible to store count information within the program relating to all sort algorithms used so these can be tabulated and compared to the mathematically derived expected performance results for all algorithms implemented. This should be carried out in a fully modular manner, through the design of appropriate functions which can be called when comparisons and moves occur, using suitable internal data and parameters.

2. Source code

This must be printed with another copy provided electronically, i.e. on floppy disk or CD-R. It must be appropriately commented, structured and indented, with suitable choices made of variable and function names etc. Care must be taken to ensure that the printing of the source code does not disguise or interfere with code layout. If possible the source code syntax should be colorised using a suitable program, (e.g. pygmentize or similar) prior to printing.

3. Test plan, data and results

As the program requirement is for a sorting algorithm test suite, the test plan must be included as part of the user interface design. Consideration must be given to the efficiency with which test inputs can be generated automatically, and test outputs efficiently and automatically confirmed correct in relation to sort sequence. Operation of automated facilities for sort correctness confirmation must be checkable using manual methods for smaller input sets. Both automated and manual approaches to obtaining results must be fully documented. If the required functionality is successfully implemented through the user interface, the test plan should describe how all the documented results can be obtained and verified through user interface actions and inputs. Your program must be capable of generating required inputs when run. Your written and printed test documentation should not include large listings of random input or sorted output, though screenshots demonstrating small manual check samples can be included for one algorithm using a small dataset or for all algorithms implemented if you have been unable to implement automated output correctness checking.

4. Report

Reports must summarise the choices of algorithms and data structures used, and critically review and analyse the software development process followed and test results obtained. Your report should analyse the expected performance of the different sorting algorithms implemented for suitable sizes of input record set sizes, and compare this with the measured performance achieved in practice. Your report should demonstrate a full analysis of all algorithms developed and tested enabling expected numbers of comparisons and moves to be derived independently of use of your program.

5. Checklist

Coursework to be handed in will comprise written report sections described above, and printed and electronic copies of all source code together with any input data files and instructions required to regenerate all test results claimed on floppy disk or CD.

Program Requirements

Your sort algorithm evaluation and test program must be able to generate useful comparative measurements of the efficiency of various sort algorithms. Measurements must include numbers of comparisons and record moves. Input data should be generated as random double precision floating point values using an approach to input generation to ensure that the greatest possible variation of possible double values between 0.0 and 100.0 which can be generated using pseudo random number generation with sufficient entropy to minimise incidence of repeated values. Students using integers or a more limited set of floating point values than achieveable will not obtain full marks. For students developing this program using 16 bit architecture compilers, test sets of the following sizes are suggested:

100, 200, 500, 1000, 2000, 5000.

For students using 32 bit achitecture compilers the following sized test input sets are suggested:

100, 400, 1000, 4000, 10000, 40000, 100000.

Students must include a test case which checks for numeric overflow in comparison and move count instrumentation variables. This may either require a change of data type to trigger numeric overflow, or a suitably large input set. Programs should automatically detect numeric overflow in instrumentation variables and terminate with a suitable error message in this event without generating incorrect counts. It is undersood that integer types for counting variables may need to be changed before and after this test case, preferably through use of a single type definition.

The following sort algorithms must be capable of being run on the same input data:

a. A quadratic iterative sort, either bubble sort or exchange sort but not both.

b. Your own implementation of the quicksort algorithm. If you don't succeed in implementing this, reduced marks will be available if you successfully integrate merge sort.

c. A tree based sort. While other tree based sorts may be implemented, Students the heap1.c program downloaded from the module website may be integrated into the student developed program.

d. A hash table, pigeon hole or bucket sort, where hash collisions are correctly handled using linked lists (chained hashing). The hash table must be dynamically allocated based on the number of items to be sorted. Some but not full marks will be awarded for less efficient collision handling approaches if data is not lost. Further marks will be lost for programs which lose collided records.

Your program should be able to apply any of the algorithms described above and chosen by the user to a set of input records generated on request from a single menu or set of interactive prompts. Results for all algorithms should be capable of being tabulated and suitably displayed to the user and be capable of being automatically tested to ensure sorted records are in ascending sorted order. Having analysed the performance expected and measured using the above algorithms, your program should display comparisons, showing percentage variations between expected and actual measured sorting performance. The expected performance must be based on an analysis of the algorithms implemented by your program clearly described in your report.