G52ADS 2014-15: Coursework THREE.
"Implementation and Analysis of Heaps"

WEIGHT: This coursework will contribute 10% of the total module score.

DEADLINE: 4pm, Friday, Nov 21, 2014.

Background & Context

Firstly you should download the supplied files, and read the README file to understand the overall structure. The overall general structure is that firstly you are given:

- PQ_IntFace.java defining an interface for a version of a Priority Queue (PQ)
- Heap2.java which gives an array based implementation of a PQ as in the lectures
- Sorts.java which uses the PQ in order to implement a HeapSort
- Main.java which is the start of an example of doing some run time scaling experiments
- HeapTest.java with some examples of how to test the Heap2

In conjunction with the lectures notes, it is highly recommended that you first ensure you understand these Java files, and so have a good understanding of binary heaps.

The binary heap described in the lectures can be directly generalised to a D-ary heap:

- A "D-ary" tree is one in which the number of children of any node is at most D
- A D-ary heap is a D-ary tree that satisfies the natural extensions of the properties of a binary heap:
    - Heap Order: the key in a parent is less than or equal to the key in any of its children
    - Complete: Each level (i.e. at each depth) is completely full except that the last level need not be full, but it must be that all nodes are as far left as possible.

Hence, a "2-ary heap" is exactly the same as the binary heap of the lectures.

Furthermore, a D-ary tree can be represented within an array by generalising the method in the lectures for a binary heap. For a node v, and with the location in the array being given by idx(v),we have, at D=2:

idx(parent(v)) = floor(idx(v) / D)

idx(child(i,v)) = D*idx(v) + i

where child(i,v) with i=0,…,D-1 are the (up to) D children of a node. You will need to find appropriate formulae for D=3 and D=4

Insertion and removal of entries follows the same overall procedure as for a binary heap, that is:

- insert: place the new entry immediately after the current last entry and then call upheap to fix its position
- removeMin: make a copy of the root so as to be able to return it; then move the last entry into the root node, and call downheap to fix its position.

The upheap and downheap methods need to be (slightly) extended so as to cope with D rather than 2 children.

Note that D-ary heaps are discussed on wiki at http://en.wikipedia.org/wiki/D-ary_heap and in many other places. You may refer to such external sources to help gain understanding, but MUST

not directly copy them (this would be plagiarism) – the essential point of the coursework is that you must demonstrate your own understanding.

Overall Description of Tasks

1. Firstly, work out for yourself how the upheap and downheap methods (given in the lectures for 2-ary heaps) need to be modified in order to handle 3-ary and 4-ary heaps
2. Starting with the partial Heap3.java and Heap4.java available from Moodle, then complete the missing portions so as obtain implementations of 3-ary and 4-ary heaps. (You should try to make the implementations run reasonably fast.)
3. Do appropriate testing to check that your implementations are functioning correctly.
4. Use the heapsort implementation and use it to do some timing tests. E.g. measure the time for sorting n integers. Do this for the 2-heap given to you, and also for the 3-ary and 4-ary that you implemented.
5. Plot a graph (maybe graphs) of the runtimes.
6. Analyse and interpret the results and graphs.

Notice that the last portions of this are (deliberately) similar to C/W 1, hence you should use your experience (and feedback from that C/W).

You need to convert your work into a report to be submitted along with your implementations of Heap3 and Heap4.

The written report should be broken down into the following parts:

A. (1.5 side maximum – was 1-side maximum, but you can use a bit more as long as the whole report stays within 3 pages!) Discussion of the algorithms used in the implementations of insert and removeMin. This should include an argument of why they correct. Also include a copy of the code of your 4-ary heap, in the given format in the submissions template and refer to it in your explanation. Also, explain what testing you did to check for correctness.
B. Figure(s) giving the experimental results for the scaling of the HeapSort as based on the 2-,3-, and 4–ary heaps.
C. (B and C together should be 2 sides maximum) Discussion of the results, and the scaling.

The questions are deliberately a bit open-ended and under-specified. It is intended to mimic the case in which you are given some candidate partially-implemented data structure and algorithm and are required to complete the implementation and say something useful about its behaviour, and how it ought to be used in practice.

Notes:

- You are not expected to produce a complete analysis!! Just something that is appropriate for the timescale of a C/W worth 10% of a 10 credit module.
- Even if you fail to implement a usable version of 3-ary and 4-ary heaps, then you can still get partial credit by doing parts B and C using just the 2-heap given to you.

Submission Requirements

You need to submit a report (both soft and hardcopy) and your source code (softcopy only).

SOFTCOPY:

By the deadline, you must submit in Moodle:

1) The electronic report. This should either be a word file or a PDF; and named g52ads-cw3.doc g52ads-cw3.docx or g52ads-cw3.pdf as appropriate.
2) Your own versions of your implementations in Heap3.java Heap4.java.
3) Your versions/extensions of Main.java or HeapTest.java that you use for creating data, testing, and doing experiments. Or any other files that you might refer to in the report.

The total length of the report section must be **no more than THREE sides of A4**.

Of course, as standard for any report, always make sure that the report starts with basic information such as your name, student ID, which coursework it is, etc.

HARDCOPY:

You MUST also submit a hardcopy of the report in the standard fashion (time-stamped with coversheet – not the coursework issue sheet - into the coursework letterbox) by the deadline. If possible, please print the hardcopy in colour (if your graph uses colour) and double-sided so as to save paper.

Remark: The double submission system is to make sure that if one is somehow lost then the other will be present (which protects you). It also gives us the option to check your code electronically, both to see it works and to check for plagiarism. As usual, you should be regularly copying all your work to the Unix servers (the H drive) as these are backed up. If you get a mark of zero and claim that your submission was lost then I will be checking the Moodle system and (possibly) the backups, but if it is not found on Moodle or the University/School fileservers system then I will have no option but continue to treat is as not submitted.

Marking Scheme

The division of marks available between the parts is roughly:

- Part A: 50%
- Parts B & C: 50%

Parts B and C are marked together, counting for ~50%, as they are so closely linked.

Marking Criteria

The main criteria for parts A and B are

1. The effectiveness and reasonableness of your experimental studies and the associated analyses. Note that this is deliberately open-ended. It is important to develop the skills of being able to decide which 'experiments' to perform, and how to analyse data. ~~Typically, you should plot at least one graph for each function.~~ (DELETED – was just an editing error)
2. The quality of writing of the report – ideally it should be both clear and brief.
3. The appropriateness and correctness, in part A, of the selected algorithms, their implementation, and their description.

Late submissions allowed for up to 7 calendar days after the deadline, but are penalised at the standard University rate (5% per working day).

Feedback

Written feedback will be given. This will both be general feedback based on all the submissions and also personalised feedback on the hardcopy, and also (later) in Moodle (Probably again including following the method of a set of keys into a table of expanded explanations.)

Plagiarism Policy

This coursework must be all your own work. You should remember that the coursework submissions will be archived, and plagiarism detection tools may be used at any time. Plagiarism is a very serious offence! Read the University regulations. If at all in doubt about whether something is allowed, then consult me or your personal tutor.

Objective

LEARNING OBJECTIVES OF THE C/W: The open-endedness is a deliberate part of the training that this C/W intends to give you. It is vital to develop the skill to decide for yourself what experiments to run, rather than just following a precise list produced by someone else. It is generally not possible to decide all experiments in advance, and so it needs to be done interactively and iteratively - if I specified the experiments then it would basically tell you the answers. This intends to help get start on the process of learning to design experiments. For example, in doing G52GRP if your program is computational intensive then you will need to design appropriate tests. If you are in industry and asked to understand the scaling of a complex piece of code then you cannot expect your boss to provide a list of the experiments to do; but will be expected to figure it out for yourself. Specifically, the objectives of this coursework are to:

- give some 'hands-on' experience with implementation of heaps and analysis of the effectiveness of various algorithm choices
- reinforce the objectives of C/W one, in terms of skills in design, implementation and analysis of data structures and algorithms

Suggestions

I will maintain a help/FAQ file online – please consult it regularly.

Released: 07-Nov-2014
Minor modifications: 16-Nov-2014
Contact: Andrew.Parkes 'at' nottingham.ac.uk