UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Algorithms — Exercises for students

Academic year 2014–2015

Lent term 2015

http://www.cl.cam.ac.uk/teaching/1415/Algorithms/
frank.stajano--algs@cl.cam.ac.uk (author of this handout)
thomas.sauerwald@cl.cam.ac.uk

Revised 2015 edition

Revision 9 of 2015-01-03 16:18:21 +0000 (Sat, 03 Jan 2015).

Introduction

From 2013–2014 I encourage all students and supervisors to use the wonderful Otter system, even though it is still somewhat experimental (feedback on your experience with it will help improve it). Otter will eventually also contain additional questions that do not appear in this document.

This document is for students and their supervisors. It contains a mix of exercises of various levels of difficulty, from the simpler ones just to check you're not reading the handout on autopilot all the way up to real exam questions. The official historical repository of exam questions is accessible from the course web page.

Students who study this course are encouraged and expected to use the CLRS3 textbook as opposed to relying only on the course handout.

Each of the recommended textbooks, and in particular CLRS3, has a copious supply of additional problems, both easier and harder than exam questions.

Students seeking clarification about these exercises are encouraged to contact their supervisor in the first instance. If this is unsuccessful, email to the lecturers about this course will be treated with higher priority if sent to the correct address listed on the front page (note that Dr Stajano's priority address contains a double hyphen).

This is a 24-lecture course with 8 supervisions, thus averaging one supervision every three lectures. Topics to be covered in supervisions are at the discretion of the supervisor but as a rough guideline they might be assigned as follows:

1. Sorting. Review of complexity and O-notation. Trivial sorting algorithms of quadratic complexity. Review of merge sort and quicksort, understanding their memory behaviour on statically allocated arrays. Heapsort.

2. Stability. Other sorting methods including sorting in linear time. Median and order statistics. Strategies for algorithm design. Dynamic programming.

3. Divide and conquer, greedy algorithms and other useful paradigms. Data structures. Primitive data structures. Abstract data types. Pointers, stacks, queues, lists, trees. Binary search trees.

4. Red-black trees. B-trees. Hash tables. Priority queues and heaps.

5. Advanced data structures. Amortized analysis: aggregate analysis, potential method. Fibonacci heaps.

6. Disjoint sets. Graph algorithms. Graph representations. Breadth-first and depth-first search. Topological sort. Minimum spanning tree. Kruskal and Prim algorithms.

7. Single-source shortest paths: Bellman-Ford and Dijkstra algorithms. All-pairs shortest paths: matrix multiplication and Johnson's algorithms.

8. Maximum flow: Ford-Fulkerson method, Max-Flow Min-Cut Theorem. Matchings in bipartite graphs. Geometric algorithms. Intersection of segments. Convex hull: Graham's scan, Jarvis's march.

Acknowledgements

Thanks to Daniel Bates, Ramana Kumar, Robin Message, Myra VanInwegen, Sebastian Funk and Wenda Li for sending corrections or suggesting better solutions to some of the exercises. If you have any more suggestions or corrections, please keep them coming.

Exercise 1
Assume that each `swap(x, y)` means three assignments (namely `tmp = x; x = y; y = tmp`). Improve the insertsort algorithm pseudocode shown in the handout to reduce the number of assignments performed in the inner loop.

Exercise 2
Provide a useful invariant for the inner loop of insertion sort, in the form of an assertion to be inserted between the "while" line and the "swap" line.

Exercise 3

$$\begin{aligned}
|sin(n)| &= O(1) \\
|sin(n)| &\neq \Theta(1) \\
200 + sin(n) &= \Theta(1) \\
123456n + 654321 &= \Theta(n) \\
2n - 7 &= O(17n^2) \\
\lg(n) &= O(n) \\
\lg(n) &\neq \Theta(n) \\
n^{100} &= O(2^n) \\
1 + 100/n &= \Theta(1)
\end{aligned}$$

For each of the above "=" lines, identify the constants k, k_1, k_2, N as appropriate. For each of the "≠" lines, show they can't possibly exist.

Exercise 4
What is the asymptotic complexity of the variant of insertsort that does fewer swaps?

Exercise 5
The proof of Assertion 1 (lower bound on exchanges) convinces us that $\Theta(n)$ exchanges are always *sufficient*. But why isn't that argument good enough to prove that they are also *necessary*?

Exercise 6
When looking for the minimum of m items, every time one of the $m-1$ comparisons fails the best-so-far minimum must be updated. Give a permutation of the numbers from 1 to 7 that, if fed to the Selection sort algorithm, maximizes the number of times that the above-mentioned comparison fails.

Exercise 7
Code up the details of the binary partitioning portion of the binary insertion sort algorithm.

Exercise 8
Prove that Bubble sort will never have to perform more than n passes of the outer loop.

Exercise 9
Can you spot any problems with the suggestion of replacing the somewhat mysterious line `a3[i3] = smallest(a1, i1, a2, i2)` with the more explicit and obvious `a3[i3] = min(a1[i1], a2[i2])`? What would be your preferred way of solving such problems? If you prefer to leave that line as it is, how would you implement the procedure `smallest` it calls? What are the trade-offs between your chosen method and any alternatives?

Exercise 10
In one line we return the same array we received from the caller, while in another we return a new array created within the mergesort subroutine. This asymmetry is suspicious. Discuss potential problems.

Exercise 11
Never mind the theoretical computer scientists, but how do you mergesort in $n/2$ space?

Exercise 12
Justify that the merging procedure just described will not overwrite any of the elements in the second half.

Exercise 13
Write pseudocode for the bottom-up mergesort.

Exercise 14
Can picking the pivot at random *really* make any difference to the expected performance? How will it affect the average case? The worst case? Discuss.

Exercise 15
Justify why running Insertion sort over the messy array produced by the truncated Quicksort might not be as stupid as it may sound at first. How should the threshold be chosen?

Exercise 16
What is the smallest number of pairwise comparisons you need to perform to find the smallest of n items?

Exercise 17
(More challenging.) And to find the *second* smallest?

Exercise 18
What are the minimum and maximum number of elements in a heap of height h?

Exercise 19
For each of the sorting algorithms seen in this course, establish whether it is stable or not.

Exercise 20
Give detailed pseudocode for the counting sort algorithm (particularly the second phase), ensuring that the overall cost stays linear. Do you need to perform any kind of precomputation of auxiliary values?

Exercise 21
Why couldn't we simply use counting sort in the first place, since the keys are integers in a known range?

Exercise 22
Leaving aside for brevity Fibonacci's original 1202 problem on the sexual activities of a pair of rabbits, the Fibonacci sequence may be more abstractly defined as follows:

$$\begin{cases} F_0 = 1 \\ F_1 = 1 \\ F_n = F_{n-2} + F_{n-1} \quad \text{for } n \geq 2 \end{cases}$$

(This yields $1, 1, 2, 3, 5, 8, 13, 21, \ldots$)
In a couple of lines in your favourite programming language, write a *recursive* program to compute F_n given n, using the definition above. And now, finally, the question: how many function calls will your recursive program perform to compute F_{10}, F_{20} and F_{30}? First, guess; then instrument your program to tell you the actual answer.

Exercise 23
Prove (an example is sufficient) that the order in which the matrix multiplications are performed may dramatically affect the total number of scalar multiplications—despite the fact that, since matrix multiplication is associative, the final matrix stays the same.

Exercise 24
Provide a small counterexample that proves that the greedy strategy of choosing the item with the highest £/kg ratio is not guaranteed to yield the optimal solution.

Exercise 25
Draw the memory layout of these two representations for a 3×5 matrix, pointing out where element (1,2) would be in each case.

Exercise 26
Show how to declare a variable of type list in the C case and then in the Java case. Show how to represent the empty list in the Java case. Check that this value (empty list) can be assigned to the variable you declared earlier.

Exercise 27
As a programmer, do you notice any uncomfortable issues with your Java definition of a list? *(Requires some thought and O-O flair.)*

Exercise 28
Draw a picture of the compact representation of a list described in the notes.

Exercise 29
Invent (or should I say "rediscover"?) a linear-time algorithm to convert an infix expression such as
`(3+12)*4 - 2`
into a postfix one without parentheses such as
`3 12 + 4 * 2 -`.
By the way, would the reverse exercise have been easier or harder?

Exercise 30
How would you deal efficiently with the case in which the keys are English words? *(There are several possible schemes of various complexity that would all make acceptable answers provided you justified your solution.)*

Exercise 31
Should the new key-value pair added by `set()` be added at the start or the end of the list? Or elsewhere?

Exercise 32
Solve the $f(n) = f(n/2) + k$ recurrence, again with the trick of setting $n = 2^m$.

Exercise 33
(Clever challenge, straight from CLRS3—exercise 12.2-4.) Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professor's claim.

Exercise 34
Why, in BSTs, does this up-and-right business find the successor? Can you sketch a proof?

Exercise 35
(Important.) Prove that, in a binary search tree, if node n has two children, then its successor has no left child.

Exercise 36
Prove that this deletion procedure, when applied to a valid binary search tree, always returns a valid binary search tree.

Exercise 37
What are the smallest and largest possible number of nodes of a red-black tree of height h, where the height is the length in edges of the longest path from root to leaf?

Exercise 38
For each of the three possible types of 2-3-4 nodes, draw an isomorphic "node cluster" made of 1, 2 or 3 red-black nodes. The node clusters you produce must:

- Have the same number of keys, incoming links and outgoing links as the corresponding 2-3-4 nodes. as the corresponding 2-3-4 nodes.
- Respect all the red-black rules when composed with other node clusters.

Exercise 39
(The following is not hard but it will take somewhat more than five minutes.) Using a soft pencil, a large piece of paper and an eraser, draw a B-tree with $t = 2$, initially empty, and insert into it the following values in order:

$$63, 16, 51, 77, 61, 43, 57, 12, 44, 72, 45, 34, 20, 7, 93, 29.$$

How many times did you insert into a node that still had room? How many node splits did you perform? What is the depth of the final tree? What is the ratio of free space to total space in the final tree?

Exercise 40
Prove that, if a key is not in a bottom node, its successor, if it exists, must be.

Exercise 41
(Trivial) Make a hash table with 8 slots and insert into it the following values:

$$15, 23, 12, 20, 19, 8, 7, 17, 10, 11.$$

Use the hash function

$$h(k) = (k \bmod 10) \bmod 8$$

and, of course, resolve collisions by chaining.

Exercise 42
Non-trivial Imagine redoing the exercise above but resolving collisions by open addressing. When you go back to the table to retrieve a certain element, if you land on a non-empty location, how can you tell whether you arrived at the location for the desired key or on one occupied by the overspill from another one? *(Hint: describe precisely the low level structure of each entry in the table.)*

Exercise 43
How can you handle deletions from an open addressing table? What are the problems of the obvious naïve approach?

Exercise 44
Prove that the sequence of trees in a binomial heap exactly matches the bits of the binary representation of the number of elements in the heap.

Exercise 45
Why do we claim that keeping the sorted-array priority queue sorted using bubble sort has linear costs? Wasn't bubble sort quadratic?

Exercise 46
Before reading ahead: what is the most efficient algorithm you can think of to merge two binary heaps? What is its complexity?

Exercise 47
Draw a binomial tree of order 4.

Exercise 48
Give proofs of each of the stated properties of binomial trees (trivial) and heaps (harder until you read the next paragraph—try before doing so).

Exercise 49
Explain with an example why, if the "peculiar constraint" were not enforced, it would be possible for a node with n descendants to have more than $O(\lg n)$ children.

Exercise 50
If we are so obsessed with keeping down the height of all these trees, why don't we just maintain all trees at height ≤ 1 all along?

Exercise 51
Build a "7-bridge" graph with this property. Then look up Euler and Königsberg and check whether your graph is or isn't isomorphic to the historical one. (Hint: you don't *have* to reconstruct the historical layout but note for your information that the river Pregel, which traversed the city of Königsberg, included two islands, which were connected by some of the 7 bridges.) Finally, build the *smallest* graph you can find that has this property.

Exercise 52
Draw in the margin an example of each of the following:

1. An anti-reflexive directed graph with 5 vertices and 7 edges.
2. A reflexive directed graph with 5 vertices and 12 edges.
3. A DAG with 8 vertices and 10 edges.
4. An undirected tree with 8 vertices and 10 edges.
5. A tree that is *not* "an undirected graph without cycles".
6. A graph without cycles that is *not* a tree.

Actually, I cheated. Some of these can't actually exist. Which ones? Why?

Exercise 53
Draw a random DAG in the margin, with 9 vertices and 9 edges, and then perform a depth-first search on it, marking each vertex with two numbers as you proceed—the discovery time (the time the vertex went grey), then a slash and the finishing time (the time it went black). Then draw the linearized DAG by arranging the vertices on a line in reverse order of their finishing time and reproducing the appropriate arrows between them. Do the arrows all go forward?

Exercise 54
Develop a proof of the correctness of the topological sort algorithm. *(Requires some thought.)*

Exercise 55
Find, by hand, a minimum spanning tree for the graph drawn in CLRS3 figure 23.4.(a) (trying not to look at nearby figures). Then see if you can find any others.

Exercise 56
Starting with fresh copies of the graph you used earlier, run these two algorithms on it by hand and see what you get. Note how, even when you reach the same end result, you may get to it via wildly different intermediate stages.

Exercise 57
Give a formal proof of the intuitive "triangle inequality"

$$v.\delta \leq u.\delta + w(u, v)$$

(where $w(u, v)$ is the weight of the edge from vertex u to vertex v) but covering also the case in which there is actually no path between s and v.

Exercise 58
Write out explicitly the elements of matrix $L^{(0)}$.

Exercise 59
To what matrix would $L^{(0)}$ map? What is the role of that matrix in the corresponding algebraic structure?

Exercise 60
Draw suitable pictures to illustrate and explain the previous comments. *Very easy; but failure to do this, or merely seeing it done by someone else, will make it unnecessarily hard to understand what follows.*

Exercise 61

- Draw a 3D picture of $\vec{p_1}, \vec{p_2}$ and $\vec{p_3}$.
- Draw a 2D picture of $\vec{p_1}, \vec{p_2}$ and the parallelogram.
- Prove that the absolute value of the determinant of the matrix $\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$, equal to $x_1y_2 - x_2y_1$, gives the magnitude of $\vec{p_3}$ and that its sign says whether $\vec{p_3}$ "comes out" of the plane ($\odot$) or "goes into" it ($\otimes$).

Exercise 62
Sketch an algorithm (not the best possible: just any vaguely reasonable algorithm) to compute the convex hull; then do a rough complexity analysis. Stop reading until you've done that. *(Requires thought.)*

Exercise 63
How can you sort a bunch of points by their polar coordinates using efficient computations that don't include divisions or trigonometry? (Hint: use cross products.) Don't read beyond this box until you've found a way.

Exercise 64
Imagine a complex CAD situation in which the set contains about a million points. Which of the two algorithms we have seen would you use to compute the convex hull if you expected it to have about 1,000 vertices? And what is the rough boundary value (for the expected number of vertices of the hull) above which you would use one algorithm instead of the other? (Is this a trick question? If so, why?)

Past exam questions

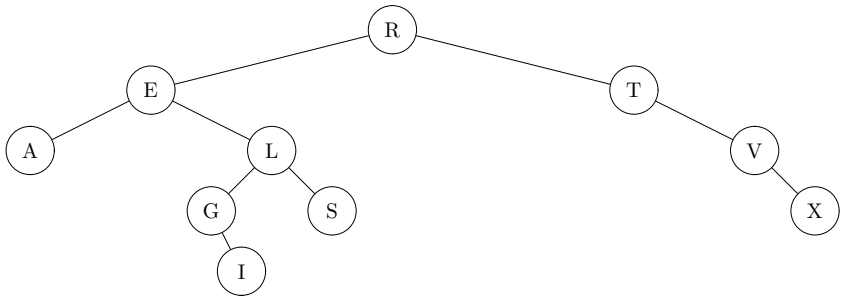
You may obtain PDFs of these and many more exam questions from the course web site. Check for exams from all the predecessor courses:

- Data structures and algorithms
- Algorithms
- Algorithms I
- Algorithms II

but also double-check, perhaps with the help of your supervisor, that the questions are covered in this year's syllabus as defined in this year's course web page.

Data Structures and Algorithms 2005–2006 — Paper 3 Question 2 (FMS)

(*a*) Briefly explain what a BST (binary search tree) is, listing its properties. Is the following binary tree a BST or not, and why? [3 marks]



(*b*) Describe an optimally efficient algorithm to find the predecessor of a given node n in a BST and explain why it works. [6 marks]

(*c*) Describe an optimally efficient algorithm for deleting a node d from a BST when neither of d's subtrees is empty. Explain why it works and prove that what remains is still a BST. [5 marks]

(*d*) Assume that node l, whose key is k_l, is a leaf of a BST and that its parent is node p, with key k_p. Prove that, of all the keys in the BST, k_p is either the smallest key greater than k_l or the largest key smaller than k_l. [6 marks]

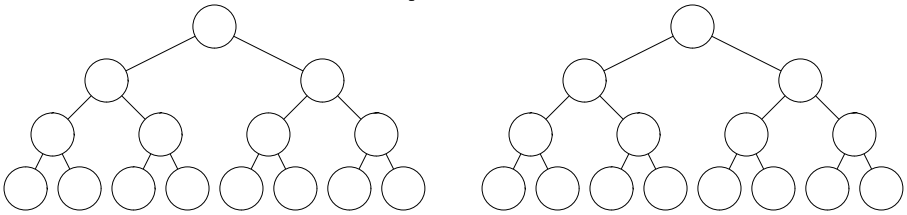
Data Structures and Algorithms 2005–2006 — Paper 6 Question 1 (FMS)

(*a*) Explain what the *heap* data structure is, state its defining properties and explain how to convert between the tree and vector representations of a heap. [2 marks]

(*b*) Describe an optimally efficient algorithm for transforming any random vector into a heap vector and explain why it works. [4 marks]

(*c*) Using the tree instead of the vector representation for clarity, apply this algorithm to the binary tree isomorphic to the letter vector "`P I S K T Z O P V N`", producing a frame by frame trace of the execution. [5 marks]

For readability, please do not draw trees any smaller than these samples, which you may use as a drawing template. Draw a new tree whenever any elements change. If you write and align your letters very neatly, it is acceptable not to draw the nodes and arcs. Hard to read traces will be penalized.



(*d*) Explain how to rearrange the heap after having extracted its top so that what remains is still a heap. Follow this procedure to extract the top three values, one by one, from the heap you built, producing a frame by frame trace as above. [5 marks]

(*e*) Describe a way to insert a new value into an existing heap in time $O(\lg n)$ where n is the heap size. [4 marks]

Data Structures and Algorithms 2006–2007 – Paper 10 Question 10 (FMS)

(*a*) Give a clear description of an efficient algorithm for finding the k-th smallest element of an n-element vector. Write some pseudocode for the algorithm and discuss its time complexity. Compare it with other plausible ways of achieving the same result. (Notes: Use zero-based indexing. You may assume for simplicity that all the elements of the vector are different.) [4 marks]

(*b*) Give a clear description of an efficient algorithm for finding the k smallest elements of a very large n-element vector. Compare its running time with that of other plausible ways of achieving the same result, including that of applying k times your solution for (*a*). (Note that in (*a*) the result of the function consists of one element, whereas here it consists of k elements. As above, you may assume for simplicity that all the elements of the vector are different.) [6 marks]

(*c*) Give an optimal algorithm for solving question (*b*) for $k = 1$. Give the worst-case number of comparisons performed by your algorithm as a function of n. (Note: exact *number of comparisons*, not just asymptotic complexity.) [4 marks]

(*d*) Same as question (*c*), but for $k = 2$. [6 marks]

Data Structures and Algorithms 2006–2007 – Paper 11 Question 9 (FMS)

(*a*) Without dwelling on the structure of the nodes and on the positional relationship between keys and subtrees (for which an example picture will be sufficient), give an otherwise complete and concise definition of a B-tree of minimum degree t, listing all the defining structural properties. [2 marks]

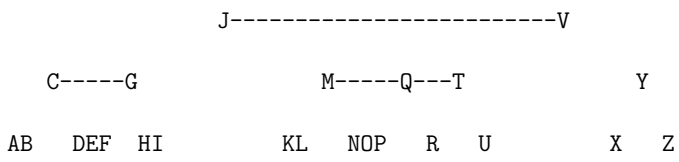
(*b*) Explain how to insert a new item into a B-tree, showing that the algorithm preserves the B-tree properties you gave in (*a*). Then insert the following values, in this order, into an initially empty B-tree whose nodes hold at most 3 keys each: `C A M B R I D G E X`. Produce a frame-by-frame "movie" in which you redraw the tree whenever it changes in any way. [4 marks]

(*c*) Let's call "bottom node" any internal node whose children are (keyless) leaves. Prove that, for any key in any node of a B-tree, either the key is in a bottom node or its successor is. You may, for simplicity, assume that all keys are distinct. [4 marks]

(*d*) Explain how to delete a value from a B-tree: [10 marks]

(*i*) Explain the overall strategy, with diagrams and pseudocode where helpful, convincing the reader that it covers all possible cases and preserves the B-tree properties.

(*ii*) Apply this procedure to the deletion of values M, Q, Y, in that order, from the following B-tree, producing a frame-by-frame movie as requested for (*b*). As before, each node holds at most 3 keys.



1

Data Structures and Algorithms 2007–2008 – Paper 10 Question 9 (FMS)

(*a*) Take an initially empty hash table with 5 slots, with hash function $h(x) = x \bmod 5$, and with collisions resolved by chaining. Draw a sketch of what happens when inserting the following sequence of keys into it: `35, 2, 18, 6, 3, 10, 8, 5`. *You are not requested to draw the intermediate stages as separate figures, nor to show all the fields of each entry in detail.* [3 marks]

(*b*) Same as question (*b*) but the hash table now has 10 slots, the hash function is $h(x) = x \bmod 10$, and collisions are resolved by linear probing. [3 marks]

(*c*) Imagine a hash table implementation where collisions are resolved by chaining but all the data stays within the slots of the original table. All entries not containing key-value pairs are marked with a boolean flag and linked together into a free list.

(*i*) Give clear explanations on how to implement the `set(key, value)` method in expected constant time, highlighting notable points and using high level pseudocode where appropriate. Make use of doubly-linked lists if necessary. [8 marks]

(*ii*) Assume the hash table has 5 slots, is initially empty and uses the hash function $h(x) = x \bmod 5$. Draw five diagrams of the hash table representing the initially empty state and then the table after the insertion of each of the following key-value pairs: `(2, A), (2, C), (12, T), (5, Z)`. In the final diagram *(and optionally in the others if it helps you—but no requirement)*, draw all the fields and pointers of all the entries. [6 marks]

Data Structures and Algorithms 2007–2008 – Paper 11 Question 7 (FMS)

Quicksort can be described as a recursive in-place sorting algorithm that performs a `partition()` operation on the given array and then invokes itself twice on two distinct subranges of the array.

(a) Describe the purpose, I/O parameters and effect of the `partition()` procedure and explain what the *pivot* is. Pseudocode not required. [3 marks]

(b) Give pseudocode for the `quicksort()` procedure that would call the `partition()` procedure you described in (a). Prove that your `quicksort()` will always terminate. [3 marks]

(c) Analyze the worst-case behaviour of Quicksort and discuss possible ways of improving it. [4 marks]

(d) Some researchers have suggested choosing the pivot from a randomly chosen location in the input array. Discuss the advantages and disadvantages of such a solution. How does it affect the worst-case and average-case behaviour? [5 marks]

(e) Define the median of an array of n numbers. Then explain clearly how to implement a `median()` procedure that would use the `partition()` procedure you described in (a). Pseudocode accepted but not required. Briefly analyze the complexity of this procedure. [5 marks]

1

Algorithms I 2008–2009 – Paper 1 Question 5 (FMS)

(*a*) State the five invariants of Red-Black Trees and briefly explain the advantages and disadvantages or Red-Black Trees over Binary Search Trees. [4 marks]

(*b*) For each of the possible types of 2-3-4-tree nodes, draw an isomorphic node cluster made of Red-Black nodes. The node clusters you produce must have the same number of keys and external links as the 2-3-4 nodes they replace and they must respect all the Red-Black tree rules when composed with other node clusters. [3 marks]

(*c*) What are the minimum and maximum possible number of nodes of a Red-Black tree with black-height h? Justify your answer. [4 marks]

(*d*) Explain, with clear pictures, what a "rotation" operation is, in the context of Binary Search Trees. [2 marks]

(*e*) Give a procedure for transforming any arbitrary n-node Binary Search Tree containing n distinct keys into any other arbitrary Binary Search Tree with the same keys. [7 marks]

Algorithms I 2008–2009 – Paper 1 Question 6 (FMS)

(*a*) State the defining properties of a min-heap. Show how to convert between the tree and the (zero-based) array representation of a min-heap. [3 marks]

(*b*) "An array sorted in ascending order is always a min-heap." True or false? If false, offer a counter-example; otherwise, prove the correctness of this statement with respect to the defining properties of a min-heap you listed in response to question (*a*). [3 marks]

(*c*) This array is not a min-heap. Why? Redraw it as a binary tree and turn it into a heap using the $O(n)$ `heapify()` procedure normally used as part of heapsort. Draw the intermediate stages as you go along and add any necessary explanations so that a reader can follow what you are doing and why. [4 marks]

```
A I E R P M S N L
```

(*d*) Perform `extractMin()` on the min-heap you produced in the previous question. As before, draw the intermediate stages and add explanations as necessary. [3 marks]

(*e*) What is the asymptotic running time of the heapsort algorithm on an array of length n that is already sorted in ascending order? Justify your answer. [3 marks]

(*f*) What is the asymptotic running time of the heapsort algorithm on an array of length n that is already sorted in *descending* order? Justify your answer. [4 marks]

1

Algorithms I 2009–2010 – Paper 1 Question 5 (RKH)

(*a*) Describe the basic principle of the *mergesort* algorithm. Illustrate your answer by showing the steps involved in sorting the array { 9, 3, 6, 2, 4, 1, 5}.
[6 marks]

(*b*) *Insertion sort* can be considered as a mergesort where each step divides an array of size n into two arrays; one of size 1 (the element to be inserted) and one of size $(n-1)$ for array length n. By solving an appropriate recurrence relation, show that this recursive version of insertion sort has a time complexity of $O(n^2)$. Assume the time complexity for merging two arrays is $O(n)$.
[5 marks]

(*c*) A programmer is tasked with sorting both arrays and linked lists. For both data structures, they intend to use the mergesort algorithm.

(*i*) Show that the time complexity of a linked list mergesort is $O(nlogn)$. Show also that the space complexity is $O(1)$, taking care to demonstrate how this can be achieved. [6 marks]

(*ii*) The programmer only knows how to merge two arrays in $O(n)$ space and linked lists in $O(1)$ space. They thus propose converting the arrays to linked lists before applying the mergesort algorithm to save on space. Comment on this strategy. [3 marks]

Algorithms I 2009–2010 – Paper 1 Question 6 (RKH)

The product of two $n \times n$ matrices, A and B is a third $n \times n$ matrix, Z, where

$$Z_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}. \tag{1}$$

(a) A programmer directly implements this formula when writing a function to multiply two matrices. Find the asymptotic time complexity of such an algorithm, taking care to justify your answer. [3 marks]

(b) An alternative strategy to compute Z is to divide A and B into four $\frac{n}{2} \times \frac{n}{2}$ matrices. Computing Z then involves eight $\frac{n}{2} \times \frac{n}{2}$ matrix multiplications followed by a series of matrix additions. This approach is then applied recursively.

(i) Identify the algorithmic strategy in use. [1 mark]

(ii) Show that the run time of this alternative strategy is given by the recurrence relation

$$t_n = K t_{\lceil f(n) \rceil} + O(g(n)) \tag{2}$$

where t_n is the time to compute the product of two $n \times n$ matrices, K is a constant you should determine, and $f(n)$ and $g(n)$ are functions that you should identify. [5 marks]

(iii) Solve the recurrence relation to find an asymptotic complexity for t_n. [5 marks]

(c) An optimisation of the algorithm presented in (b) means that only seven matrix multiplications are needed rather than the eight previously suggested. State the new recurrence relation and solve it to show that this algorithm is $O(n^{log_2 7})$. [5 marks]

Algorithms I 2010–2011 – Paper 1 Question 5 (FMS)

Mathematical hint: the following series converges to the indicated value if $|x| < 1$:

$$\sum_{m=1}^{\infty} mx^m = \frac{x}{(1-x)^2}.$$

(a) The binary min-heap provides a `decreaseKey()` method that, when applied to an element, decreases its key while preserving the properties of the data structure.

(i) Give a brief and clear description of how `decreaseKey()` works. [3 marks]

(ii) The standard `decreaseKey()` only accepts a positive argument. Describe an implementation that removes that restriction, that is to say a heap method that can move the key value up as well as down, as specified by the sign of its argument. [4 marks]

(b) Generalize the *binary* min-heap to one where nodes have not 2 but k children.

(i) State the two defining properties of a min-heap, one constraining the shape and one constraining the keys of the data structure, and describe how to represent a k-ary min-heap as an array. [4 marks]

(ii) Give a clear description of an algorithm (a simple generalization of the well-known one for binary heaps) that takes an arbitrary n-item array and efficiently rearranges its elements to turn it into an array representing a k-ary heap. [4 marks]

(iii) Analyze its complexity as a function of n and k. [5 marks]

Algorithms I 2010–2011 – Paper 1 Question 6 (FMS)

Mathematical hint: the following series converges to the indicated value if $|x| < 1$:

$$\sum_{m=1}^{\infty} m x^m = \frac{x}{(1-x)^2}.$$

(a) You are given an unsorted array of n items of which you must return the top k, in any order. Give a clear and accurate description of two efficient algorithms for solving the problem, following the hints below, and derive their time complexity in terms of n and k.

(i) For this attempt, use a priority queue. [2 marks]

(ii) For this attempt, start by finding the k-th largest item, as when computing order statistics. Describe all the steps in detail. [8 marks]

(b) You are given a hash table that stores n keys. It has m slots, collisions are resolved by chaining (each hash table slot contains a pointer to the head of the corresponding chain; no elements are stored in the table itself) and no chain is longer than L elements. Each slot has an extra field giving the length of the corresponding chain. Give a clear and accurate description of an efficient algorithm for returning a random key from the table, such that each key has equal probability of being selected, and analyze its time complexity. Assume the availability of a function `random(x)` that returns in constant time a random integer between 0 and x. The time complexity of a reasonable algorithm will not exceed $O(m + L)$, but solutions awarded full marks will improve on that. [10 marks]

1