UNIVERSITY OF CAMBRIDGE

Department of Computer Science and Technology

Algorithms— Example sheets

Professor Frank Stajano

Computer Science Tripos — Part IA

Academic year 2020–2021

Lent term 2021

Lecture recordings:
http://frankstajanoexplains.com
Course web page:
http://www.cl.cam.ac.uk/teaching/2021/Algorithms/

Email about this course, from a `@cam.ac.uk` address,
to the following special address, will be treated with higher priority:

`frank.stajano--algs2021@cl.cam.ac.uk`

Revised 2021 edition

Revision 29 of 2021-01-04 03:37:23 +0000 (Mon, 04 Jan 2021).

Introduction

This document contains a mix of exercises of various levels of difficulty, from the many simple ones just to check you're not reading the handout on autopilot all the way up to real exam questions.

If you want to challenge yourself with even more exam questions, visit https://www.cl.cam.ac.uk/teaching/exams/pastpapers/: look for anything you like in any of Algorithms, Algorithms I, Algorithms II, Data Structures and Algorithms, but be aware that some past questions may refer to topics that are no longer taught in the Computer Science Tripos, so you may have to learn new stuff on your own (and well done if you do). The questions in this curated example sheet, instead, are all covered in this year's syllabus.

I encourage and expect you to use the CLRS3 textbook as opposed to relying only on the course handout. I also very strongly encourage you to program all these algorithms and data structures by yourself, with textbook and notes and web browser closed, and test them extensively. There is really no substitute for this. You can't learn to play the piano by going to lectures or reading books if you don't also practice a lot, and writing correct and beautiful code isn't any different.

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

If you seek clarification about these exercises, please contact your supervisor in the first instance. If your supervisor cannot help, emails to me about this course will be treated with higher priority if they are sent to the correct address listed on the front page (note that my priority address contains two consecutive hyphens) and if they demonstrate you had a serious go at the problem, both by yourself and with your supervisor.

This is a 24-lecture course, of which I now only teach the first half. The current recommendation from the department is to allocate a supervision every four lectures. Topics to be covered in supervisions are at the discretion of the supervisor but as a rough guideline I have split the material for the first half of the course into three example sheets, each of which should be covered in *about* 4 lectures (but no promises). Supervisors are still free to rebalance topics as they see fit, for example by bringing some Binary Search Tree exercises from the third supervision (which covers many data structures) into the second (which by comparison is less loaded)[1]. Students, do not be afraid to read ahead slightly if your supervisor gave you a problem on a topic I have not yet lectured by the

[1]Supervisors: should you wish to do that, suitable exercises are 34–37 and suitable exam questions are those marked with BST in Example Sheet 3.

time your supervision takes place.

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. Stability. Other sorting methods including sorting in linear time. Median and order statistics.

2. Strategies for algorithm design. Dynamic programming. Divide and conquer, greedy algorithms and other useful paradigms. Data structures. Primitive data structures. Abstract data types. Pointers, stacks, queues, lists, trees.

3. Binary search trees. Red-black trees. B-trees. Hash tables. Priority queues and heaps.

Acknowledgements

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

Example sheet 1

Covering lectures 1–4 approximately.

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. Stability. Other sorting methods including sorting in linear time. Median and order statistics.

Exercise 0
If you were to compare every possible subsequence of the first string to every possible subsequence of the second string, how many comparisons would you need to perform, if the lengths of the two strings were respectively m and n?

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 insertsort, in the form of an assertion to be inserted between the "while" line and the "swap" line.

Exercise 3
Write down an incorrect definition for $o(n)$ by taking the definition of $O(n)$ and replacing $\leq$ by $<$. Then find values for k and N that, by this definition, would allow us to claim that $f(3n^2) \in o(n^2)$.

Exercise 4

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

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

Exercise 5

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

Exercise 6

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 7

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 selectsort algorithm, maximizes the number of times that the above-mentioned comparison fails.

Exercise 8
Code up the details of the binary partitioning portion of the binary insertsort algorithm.

Exercise 9
Consider the smallest ("lightest") and largest ("heaviest") key in the input. If they both start halfway through the array, will they take the same time to reach their final position or will one be faster? In the latter case, which one, and why?

Exercise 10
Prove that bubblesort will never have to perform more than n passes of the outer loop.

Exercise 11
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 12
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 13
Never mind the theoretical computer scientists, but how do you mergesort using a workspace of size not exceeding $n/2$?

Exercise 14
Justify that the merging procedure just described will not overwrite any of the elements in the second half.

Exercise 15
Write pseudocode for the bottom-up mergesort.

Exercise 16
What are the minimum and maximum number of elements in a heap of height h?

Exercise 17
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 18
Justify why running insertsort (a quadratic algorithm) 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 19
What is the smallest number of pairwise comparisons you need to perform to find the smallest of n items?

Exercise 20
(More challenging.) And to find the *second* smallest?

Exercise 21
For each of the sorting algorithms seen in this course, establish whether it is stable or not.

Exercise 22
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 23
Why couldn't we simply use counting sort in the first place, since the keys are integers in a known range?

See also the following exam questions (all clickable hyperlinks for your convenience, at least so long as the URLs don't change):

y2020-p01-q08
y2018-p01-q07
y2016-p01-q08 (a), (d)
y2014-p01-q08 (a), (b)
y2014-p01-q07
y2012-p01-q05
y2011-p01-q05
y2010-p01-q05
y2008-p11-q07
y2007-p10-q10
y2007-p01-q11
y2007-p01-q04 (a), (b), (d)
y2006-p06-q01
y2006-p01-q12 (b)
y2006-p01-q11
y2006-p01-q04

Example sheet 2

Covering lectures 5–8 approximately.

Strategies for algorithm design. Dynamic programming. Divide and conquer, greedy algorithms and other useful paradigms. Data structures. Primitive data structures. Abstract data types. Pointers, stacks, queues, lists, trees.

Exercise 24
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 25
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 26
There could be multiple distinct longest common subsequences, all of the same length. How is that reflected in the above algorithm? And how could we generate them all?

Exercise 27
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 28
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 29
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 30
As a programmer, do you notice any uncomfortable issues with your Java definition of a list? *(Requires some thought and O-O flair.)* Hint: a sensible list class should allow us to invoke some kind of `isEmpty()` method on any object of type list, even if (particularly if) it represents an empty list. Does your definition allow this? If not, be uncomfortable. If yes, find something else about it to be uncomfortable about `;-)`

Exercise 31
Draw a picture of the compact representation of a list described in the notes.

Exercise 32
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 33
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 34
Should the new key-value pair added by `set()` be added at the start or the end of the list? Or elsewhere?

Exercise 35
Solve the $f(n) = f(n/2) + k$ recurrence, again with the trick of setting $n = 2^m$.

See also the following exam questions:
y2020-p01-q07
y2019-p01-q07
y2016-p01-q08 (b), (c)
y2016-p01-q07
y2015-p01-q08
y2013-p01-q06
y2006-p03-q02

Example sheet 3

Covering lectures 9–12 approximately.

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

Exercise 36
(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 37
Why, in BSTs, does this up-and-right business find the successor? Can you sketch a proof?

Exercise 38
(Important.) Prove that, in a binary search tree, if node n has two children, then its successor has no left child.

Exercise 39
Prove that this deletion procedure, when applied to a valid binary search tree, always returns a valid binary search tree.

Exercise 40
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 41
With reference to the rotation diagram in the handout, and to the stupid way of referring to rotations that we don't like, what would a *left* rotation of the D node be instead? (Hint: it would *not* be the one marked as "Left rotation" in the diagram.)

Exercise 42
During RBT insertion, if p is red and g is black, how could u ever possibly be black? How could p and u ever be of different colours? Would that not be an immediate violation of invariant 5?

Exercise 43
Draw the three cases by yourself and recreate, without reading, the correct procedure to fix each of them. Then apply it to figure 13.4.(a) of CLRS3, without looking at the rest of the figure.

Exercise 44
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 45
(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 46
Prove that, if a key is not in a bottom node, its successor, if it exists, must be.

Exercise 47
(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 48
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 49
How can you handle deletions from an open addressing table? What are the problems of the obvious naïve approach?

Exercise 50
Why do we claim that keeping the sorted-array priority queue sorted using bubblesort has linear costs? Wasn't bubble sort quadratic?

Exercise 51
Before reading ahead: what is the most efficient algorithm you can think of to merge two binary heaps? What is its complexity?

Exercise 52
Draw a binomial tree of order 4.

Exercise 53
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 54
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.

See also the following exam questions:
y2019-p01-q08 (a) BST, (b−−e)
y2018-p01-q09 (a), (b)
y2018-p01-q08
y2017-p01-q08
y2017-p01-q07 BST
y2015-p01-q07
y2014-p01-q08 (c), (d) BST
y2012-p01-q06 BST
y2009-p01-q05 BST
y2009-p01-q06
y2008-p10-q09

y2008-p01-q11 BST
y2008-p01-q04
y2007-p11-q09
y2007-p01-q12 BST