Algorithms and Data Structures (BADS)

Exam 31 May 2013

Thore Husfeldt, ITU

Instructions

What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper you can do so. (It won't help.)

You can't bring electronic aids (such as a laptop) or communication devices (such as a mobile phone). If you really want, you can bring an old-fashioned pocket calculator (not one that solves recurrence relations), but I can't see how that would be of any use to you.

Answering multiple-choice questions. In the multiple-choice questions, there is one and only one correct answer. However, to demonstrate partial knowledge, you are allowed to check 2 or more boxes, but this earns you less than full points for that question.

number of checked boxes	0	1	2	3	4
points if correct answer checked		1	0.5	0.21	0
points if correct answer not checked	0	−0.33	−0.5	−0.62	

In particular, the best thing is to only check the correct answer, and the worst thing is to check all answers but the correct one. If you don't check anything (or check *all* boxes) your score is 0. Also, if you check boxes at random, your expected score is 0. Some questions are worth more points, or allow more than four choices; their points are scaled accordingly. For more details, read [Gudmund Skovbjerg Frandsen, Michael I. Schwartzbach: A singular choice for multiple choice. SIGCSE Bulletin 38(4): 34–38 (2006)].

(Just to make sure: a question that is not multiple-choice cannot give you negative points.)

Where to write. Please try to answer the exam by writing directly on the exam set. If you run out of space, or change your mind, then of course you can answer questions of a separate piece of paper. Just make it very clear (cross out everything and write "see separate paper, page 1" or something like that.) Question 4b needs to be answered on a separate paper anyway.

Typographic remark. I follow the typographic convention used impliclty in the course book that a one-letter Java variable, such as `N`, is typeset in italics like a mathematical variable in body text: N.

Base revision 96e1eef. . ., Sun May 28 17:57:45 2017 +0200, Riko Jacob (@ITU Air 13).

1. Analysis of algorithms

(a) (*1 pt.*) Which pair of functions satisfy $f(N) \sim g(N)$?

A $(N+1)(N+N)$ and $2N$ B $(N+1)(N+N)$ and N^3
C $\log N + \log 3N$ and $3 \log N$ [✓ D] 2^N and $2^N + N^2$

(b) (*1 pt.*) How many array accesses does the following piece of code perform?

```
for (int i = 0; i < N; i = i+1)
    for (int j = 1; j < N; j = j*2)
        A[i] = j;
```

A $\sim N^2$ B $\sim \frac{1}{2}N^2$ [✓ C] $\sim N \log N$ D $\sim N^{1/2}$

(c) (*1 pt.*) What is the asymptotic running time of the following piece of code?

```
if (N % 2 == 0) // N is even
  for (int i = 0; i < N; i = i+1)
      for (int j = 0; j < N; j = j+1)
         A[i] = j;
else // N is odd
  for (int i = 0; i < N; i = i+1)
      A[i] = i;
```

A $O(N)$ B $O(N\sqrt{N})$ C $O(N \log N)$ [✓ D] $O(N^2)$

(d) (*1 pt.*) Find a recurrence relation for the number of multiplications performed by the following recursive method:

```
static int f(int N)
{
    if (N > 1) return 2*f(N - 1);
    else return 3;
}
```

(Choose the smallest correct estimate.)

A $T(N) = T(N-1) + 2$ B $T(N) = T(N-1)$
C $T(N) = 2 \cdot T(N-1)$ [✓ D] $T(N) = 2 \cdot T(N-1) + 1$

(e) (*1 pt.*) Assume I have a method `f(int K)` that takes constant amortized time per call, but linear time (in K) in the worst case. Consider the following piece of code:

```
for (int i = 1; i < N ; i = i+1)
    f(i);
```

What is the running time of this piece of code as a function of N?

[✓ A] Amortized linear, but quadratic in the worst case.

B Worst case linear.

C Amortized linear, but worst case linearithmic.

D Amortized linear, and we don't have enough information about `f` to give a worst case bound.

```java
public class Z<Key extends Comparable<Key>, Value> {
    Node first;  // first node in  linked list

    // a helper linked list data type
    class Node {
        Key key;
        Value val;
        Node next;

        public Node(Key key, Value val, Node next)  {
            this.key  = key;
            this.val  = val;
            this.next = next;
        }
    }

    public Value get(Key key) {
        for (Node x = first; x != null; x = x.next) {
            if (key.equals(x.key)) return x.val;
        }
        return null;
    }

    public final void put(Key key, Value val) {
        for (Node x = first; x != null; x = x.next)
            if (key.equals(x.key)) { x.val = val; return; }
        first = new Node(key, val, first);
    }
}
```

Figure 1: Class Z.

2. Class Z. The next few questions all concern the class defined in fig. 1.

(a) (*1 pt.*) Class Z behaves like which well-known data structure?

A Stack. B Bag. C Priority queue. [✓ D] Symbol table.

(b) (*1 pt.*) Draw the data structure (including all `Node`s) after the operations `put("1",3), put("5",4), put("1",5)`.

(c) (*1 pt.*) Assume there are N items in the data structure. How many key comparisons (i.e., calls to `key.equals()`) does a call to `get()` take in the worst case?

[✓ A] $\sim N$. B $O(\log N)$. C $O(N^2)$. D $\sim \frac{1}{2}N$.

(d) (*1 pt.*) What is the total running time of the following sequence of N operations: `put("1",1)`, `put("1",2)`, ..., `put("1",`N`)`?

A constant. [✓ B] linear. C linearithmic. D quadratic.

(e) (*1 pt.*) What is the total running time of the following sequence of N operations: `put("1",1)`, `put("2",1)`, ..., `put("`N`",1)`?

A constant. B linear. C linearithmic. [✓ D] quadratic.

(f) (*2 pt.*) Modify `Z` with a method `size()` that returns the number of items stored in the data structure. The method has to work in constant time. Your answer goes into the gray box. Write complete and correct Java code, and refer to the line numbers in figure 1. (For example, write "between line 28 and 29: `return val`".)

(g) (*2 pt.*) (Same question as above, but with inheritance.) Write a class `S extends Z` with a method `size()` that returns the number of items stored in the data structure. Your new class `S` extends the original class `Z` from fig. 1. (Note that the `put` method is `final` in `Z`, so you cannot override it.) The method has to work in linear time in the size.

This means "at the end of all 3 operations," not "after each operation."

```
class S<Key extends Comparable<Key>, Value>  extends Z<Key, Value>
{
    public int size()
    {






    }
}
```

3. Operation of common algorithms and data structures.

(a) (*1 pt.*) Consider the following sequence of operations on a stack:

```
push(2)  push(8)  pop()  pop()  push(4)  push(5)  pop()  push(6)  pop()
```

What sequence of values is returned by the pop-operations? (Your answer goes into the shaded box:)

(b) (*1 pt.*) I have sorted the word "niroht iro iron irod rubmob rufob rufib nilab nilawd niolg nio ilik ilif" using MSD string sort ([SW, section 5.1]). In the figure below, mark all characters that were examined by the MSD string sort with a circle.

```
ilif
ilik
iro
irod
iron
nilab
nilawd
nio
niolg
niroht
rubmob
rufib
rufob
```

(c) (*1 pt.*) Draw the 2-3 tree that results when you insert the keys A N N O Y I N G in that order into an initially empty tree. (Your answer goes into the shaded box:)

In the following 4 questions, consider the strings,

`oin gloin bombur bifur bofur fili kili ori`

as input to a sorting algorithm. Each question describes an intermediate stage of one and only one sorting algorithm: quicksort, (top-down) merge sort, selection sort, and MSD string sort. (Every algorithm corresponds to exactly one sequence.) Which is which?

(d) ($\frac{1}{2}$ *pt.*) `bombur bifur bofur fili gloin kili oin ori`

A quick B merge C select [✓ D] MSD

(e) ($\frac{1}{2}$ *pt.*) `bofur bifur fili bombur gloin ori kili oin`

[✓ A] quick B merge C select D MSD

(f) ($\frac{1}{2}$ *pt.*) `bifur bombur gloin oin bofur fili kili ori`

A quick [✓ B] merge C select D MSD

(g) ($\frac{1}{2}$ *pt.*) `bifur bofur bombur oin gloin fili kili ori`

A quick B merge [✓ C] select D MSD

(h) (*1 pt.*) Our analysis of hash tables depends on the uniform hashing assumption (Assumption J in section 3.4 of [SW]). When would this assumption *fail*?

[✓ A] The hash function was badly chosen for the keys in question.

B Many keys are the same, or very similar.

C The hash table was chosen too small.

D The hash function cannot be computed efficiently.

(i) (*1 pt.*) Continuing the above question, assume I implemented hashing with separate chaining using a hash function that does *not* satisfy the uniform hashing assumption. What would happen?

[✓ A] `get` and `put` no longer work in expected constant time.

B The data structure fills up and stops working.

C Space usage increases dramatically.

D The running times for the operations becomes those of a balanced binary search tree.

(j) (*1 pt.*) Consider the keys `iro iron irod rubmob rufob rufib niolg nio` with associated values $1, 2, \ldots, 8$, respectively. (For instance, `irod` has value 3.) Insert them into an initally empty trie and draw the result:

4. Design of algorithms.

The *sentence infection* problem is as follows. You are given a long text, such as a book. The text can be broken down into sentences, and every sentence into words. Here's an example with 5 sentences.

> In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled with ends of worms and an oozy smell. Nor yet a dry, bare, sandy hole with nothing in it to sit down on or to eat. It was a hobbit-home, and that means comfort. Its perfectly round door had been painted green.

A sentence is *infected* if it contains the word 'nasty.' In our example, sentence 2 is infected. Infection *spreads* through the text in the sense all words in an infected sentence become infected themselves, and thereby infect other sentences. In our example, sentence 1 is infected from sentence 2 via 'hole'. So is sentence 3. In sentence 3, the word 'it' becomes infected, so the process spreads to sentence 4. The process continues as far as it can. (Let's just agree that case does not matter, so 'nasty' infects 'Nasty'.)

(a) (*0 pt.*) Convince yourself that the 5th sentence remains uninfected.

(b) (*5 pt.*) Design an algorithm for this problem. It has to be fast enough to handle a good-sized book (megabytes of input). The input is given as a `String` (you can assume we have enough memory to store the whole text), and the output has to be the uninfected sentences (their order is not important). To fix notation, let's say there are n_1 characters in the total input, n_2 sentences, and n_3 words (on average) per sentence.
You are encouraged to make use of existing algorithms, models, or data structures from the book, but please be precise in your references (for example, use page numbers or full class names of constructions in the book). Estimate the running time of your solution. Be short and precise. This question can be perfectly answered on half a page of text. If you find yourself writing much more than one page, you're using the wrong level of detail. However, it is a very good idea to include a drawing of a concrete (small) example. If you can avoid it, please do not write code. (However, some people have an easier time expressing themselves clearly by writing code. In that case, go ahead.)