Algorithms and Data Structures (BADS)

Exam 28 May 2014

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.)

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 4a 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 c5a2536. . ., Mon Jan 30 13:44:50 2017 +0100, Riko Jacob (@ITU Air 13).

1. Analysis of algorithms

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

$\boxed{A}$ $(N+1)(N+N+N)$ and $2N^2$ $\boxed{B}$ $N+17$ and N
$\boxed{C}$ $\log 3N$ and $3\log N$ $\boxed{D}$ $(N\log N)+16N$ and $(2N\log N)+16N$

(b) (*1 pt.*) How many stars are printed?

```
for (int i = N; i > 1; i = i/2) StdOut.print("*");
```

$\boxed{A}$ $\sim \log N$ $\boxed{B}$ $\sim N$ $\boxed{C}$ $\sim N\log N$ $\boxed{D}$ $\sim \frac{1}{2}N^2$

(c) (*1 pt.*) How many stars are printed when I call $f(N)$?

```
static void f(int K)
{  for (int i = 0; i < K; i = i+1) g(i); }

static void g(int K)
{  for (int i = 0; i < K; i = i+1) StdOut.print("*"); }
```

$\boxed{A}$ logarithmic in N $\boxed{B}$ linear in N $\boxed{C}$ linearithmic in N $\boxed{D}$ quadratic in N

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

```
if (N < 1000)
  for (int i = 0; i < N; i = i+1)
      for (int j = 0; j < N; j = j+1)
          A[i] = j;
else
  for (int i = 0; i < N; i = i+1)
      A[i] = i;
```

$\boxed{A}$ logarithmic in N $\boxed{B}$ linear in N $\boxed{C}$ linearithmic in N $\boxed{D}$ quadratic in N

(e) (*1 pt.*) Find a recurrence relation for the number of arithmitic operations (additions and subtractions) performed by the following recursive method:

```
static int fib(int N)
{
    if (N >= 2) return fib(N-1) + fib(N-2);
    if (N == 1) return 1;
    if (N == 0) return 1;
}
```

(Choose the smallest correct estimate.)

$\boxed{A}$ $T(N) = T(N-1)+3$ $\boxed{B}$ $T(N) = T(N-1)+T(N-2)$
$\boxed{C}$ $T(N) = T(N-1)+1$ $\boxed{D}$ $T(N) = T(N-1)+T(N-2)+3$

(f) (*1 pt.*) Assume I have a method `f(int K)` that I call inside a for-loop like this:

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

I want the entire code to run in time linear in N. How can I achieve this?

$\boxed{A}$ f must be a hash function.

$\boxed{B}$ This is impossible.

```
public class A<Key, Value>
{
  private Key[]   keys;
  private Value[] vals;
  private int N;

  public A(int capacity)
  {
    keys = (Key   []) new Object[capacity];
    vals = (Value []) new Object[capacity];
    N = 0;
  }

  public Value get(Key key)
  {
    for (int i = 0; i < N; i++)
      if (keys[i].equals(key))
        return vals[i];
    return null;
  }

  public void put(Key key, Value val)
  {
    for (int i = 0; i < N; i++)
      if (keys[i].equals(key))
      { vals[i] = val; return; }
    keys[N] = key;
    vals[N] = val;
    N += 1;
  }
}
```

Figure 1: Class A.

C f must run in constant amortized time, but may run in linear worst-case time in K.

D f must run in worst-case time logarithmic in K.

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

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

A Stack.
B Bag.
C Priority queue.
D Symbol table.

(b) (*1 pt.*) Draw the data structure after the following operations: (This means "*at the end* of the operations," not "*after each* operation." Make sure you draw the entire data structure. Make sure to include all instance variables.)

```
A a = new A<String, Integer>(5);
a.put("A",13);
```

```
a.put("B",14);
a.put("B",15);
```

(c) (*1 pt.*) What is the *total* running time of the following code (N is an int variable)

```
A a = new A<Integer, Integer>(N);
for (int i = 0; i < N; i++) a.put(i,0);
```

A $O(\log N)$. B $O(N)$.
C $O(N \log N)$. D $O(N^2)$.

(d) (*1 pt.*) What is the *total* running time of the following code (N has type `int`)

```
A a = new A<Integer, Integer>(N);
for (int i = 0; i < N; i++) a.put(0,i);
```

A $O(\log N)$. B $O(N)$.
C $O(N \log N)$. D $O(N^2)$.

(e) (*1 pt.*) In the previous question, assume I initialised the data structure with a smaller value for `capacity`, like this:

```
A a = new A<Integer, Integer>(N/2);
for (int i = 0; i < N; i++) a.put(0,i);
```

What happens when I run this code?

A The data structure uses half as much space.

B The running time for `put` is halved.

C `ArrayIndexOutOfBoundsException`.

D `OutOfMemoryError`.

(f) (*1 pt.*) In this question (and only here), assume `Key implements Comparable`. How could I modify `class A` to significantly reduce the running time for `get` (possibly spending more time for `put`).

A View the keys as priorities in a priority queue.

B Use a linked list instead of an array.

C Keep the keys in sorted order.

D Implement *array resizing* ([SW] pp. 136).

(g) (*1 pt.*) Modify `class A` to support the method `public int size()`, which reports the number of (non-*null*) values in the data structure. Your method must run in constant time. Write correct Java code and refer to the line numbers in figure 1. For instance, write "insert after line 24: `int K = 5;`".

(h) (*1 pt.*) Modify `class A` to handle arbitrarily large `N`, no matter the initial capacity. *Hint*: use array resizing.

(i) (*1 pt.*) With your modification from the previous question, what is the total running time of the following piece of code:

```
A a = new A<Integer, Integer>(1);
for (int i = 0; i < N; i++) a.put(i,i);
```

A $O(\log N)$. B $O(N)$.
C $O(N \log N)$. D $O(N^2)$.

(j) (*2 pt.*) Using your modification from question h, what is the amortized running time of `get` and `put`, starting from an empty data structure? Briefly explain your answer; in particular mention if the amortized running time asymptotically differs from the worst case running time, and if yes, how. (Don't write more than a few sentences.)

3. Operation of common algorithms and data structures.

(a) (*1 pt.*) Consider the following sequence of operations on a data structure, where a number i means `insert(`i`)` and "$*$" means `remove()`. The data structure is initially empty.

$$1 \quad 12 \quad 5 \quad * \quad 3 \quad 7 \quad * \quad * \quad * \quad 2 \quad 4 \quad 13 \quad * \quad 14 \quad 15 \quad * \quad * \quad *$$

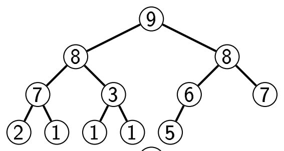
The output resulting from these operations is:

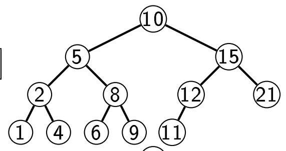
$$1 \quad 3 \quad 5 \quad 7 \quad 2 \quad 4 \quad 12 \quad 13$$

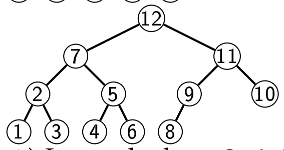
What is the data structure?

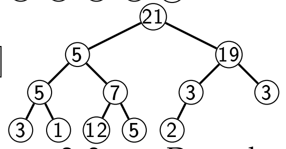
A A bag B A stack
C A FIFO queue D A priority queue

(b) (*1 pt.*) Which of the following trees is in heap order?

A


B


C


D


(c) (*1 pt.*) Insert the keys `2 4 5 1 3 6 7` in that order into a 2–3 tree. Draw the resulting tree.

(d) (1 *pt.*) Run (i) insertion sort and (ii) selection sort on the 8-letter input

```
G O D Z I L L A
```

and stop after exactly 3 calls to `exch`.
Write the resulting sequences. (Your answer consist of two sequences of exactly 8 characters, the first is for insertion sort, the second for selection sort.)

Insertion sort:

Selection sort:

(e) (*1 pt.*) Consider the key–value pairs

key	E	X	A	M	Q	U	E	S	T	I	O	N
value	0	1	2	3	4	5	6	7	8	9	10	11
hash	4	3	0	2	1	0	4	3	4	3	4	3

We use the hash function `(key.hashCode() & 0x7fffffff) % 7`. To spare you the calculations, the hash values are given in the table above as well. The elements are inserted from left to right into an initally empty hash table using separate chaining. Draw the result in the style of the book [SW, p. 464]:

(f) (*1 pt.*) In which sense is a balanced binary search tree better than a binary search tree?

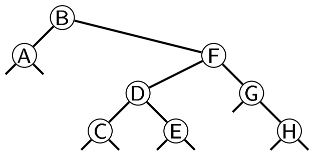
A It uses less space because nodes are distributed more evenly in memory.

B It is easier to implement than a BST, when using the red–black representation .

C The worst-case performance is better.

D They don't require the `Keys` to `implement Comparable`.

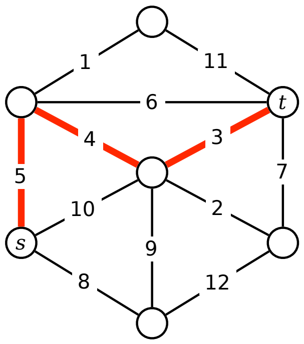
(g) (*1 pt.*) Consider this `BST` [SW, sec. 3.2]:



Delete `F` from it using the book's `class BST` (algorithm 3.3) and draw the resulting tree.

4. Design of algorithms

Consider a weighted, undirected graph, like this one:



The weights are *security levels*. To travel along an edge of security level w you need security clearance $\geq w$. In the above example, you can travel from s to t if you have security clearance 5 or higher. (Walk along the highlighted path.)
To fix notation, there are V vertices and E edges. The weights are taken from the set $\{1, 2, \ldots, W\}$ for some fixed integer W; the value of W is polynomial in V.

(a) (*5 pt.*) The input is a weighted graph G as above, including two vertices called s and t.
Design an algorithm that outputs the minimum $c \in \{1, 2, \ldots, W\}$ such that you can get from s to t using security clearance c.
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. (Even less, in fact.) 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.)
You are evaluated on correctness and efficiency of your solution, and clarity of explanation.