Algorithms and Data Structures (BADS)

Exam 31 May 2011

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	.5	0.21	0
points if correct answer not checked	0	-0.33	-.5	-0.74	

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. 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.)

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 04e2f12. . ., Tue Jan 24 16:54:59 2017 +0100, Thore Husfeldt.

Analysis of algorithms

1.

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

A $(N^4+1)(N+N^3)$ and $2N^7$

B $(N \log_2 N)(N+7)$ and N^2

C $8N^2+3N$ and $8N^2+160 \log N$

D 2^N and 2^{2N}

(b) (*1 pt.*) Find a recurrence relation for the number of arithmetic operations (subtractions, additions, multiplications, divisions) performed by the following recursive method:

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

(Choose the smallest correct estimate.)

A $T(N) = T(N-1) + T(N-2) + 1$

B $T(N) = T(N-1) + T(N-2) + 3$

C $T(N) = 3T(N-1) + 3T(N-2)$

D $T(N) = 3T(N-1) + 3T(N-1) + 1$

(c) (*1 pt.*) Assume you have a data structure that maintains a set `S` under insertion. Assume that the operation "`S.insert(key)`" takes logarithmic time in the number of elements in `S`.
Determine the order of growth of the running time of the following piece of code, starting with an empty set `S`.

```
for (int i = 0; i < N ; i++){
    S.insert(i);
    for (int j = 0; j < 10; j++)
        StdOut.println(j);
    }
```

A logaritmic time

B linear time

C linearithmic time

D quadratic time

(d) (*1 pt.*) I've performed a doubling test on my program `MyCode`, running it random instances of increasing size (see [SW, p. 144]). Here's the output:

```
% java MyCode
   10   0.0
   20   0.0
   40   0.2
   80   1.6
  160  12.9
  320 102.5
```

Which hypothesis can I *reject* on the basis is of this data?

A The order of growth of the running time is N^2.

B The program uses $3N^2$ comparisons.

C The running time is $\sim aN^3$ for some a.

D The program uses $4N^3 + N^2$ array accesses.

Class X

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

2.

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

A Stack. B Queue.
C Priority queue. D Union–Find.

(b) (*1 pt.*) Write the body of a method `int size()` that returns the number of elements in the data structure.

A `return N;` B `return A.length;`
C `return A[N];` D `return A.length - N;`

(c) (*1 pt.*) Which invariant does the data structure maintain after every public operation?

A `N < A.length` B `A[max] >= A[i]`
C The last inserted element is at `A[n-1]` D `A.length/2 <= N < A.length`

(d) (*1 pt.*) How many array accesses does a single call to `X.remove` take? (To make this well-defined we assume that the compiler performs no clever optimisations. That is, every array access we've written in the code will actually be performed.)

A $\sim 6N$. B 2.
C $\sim 2N$. D 7.

(e) (*1 pt.*) Consider the class in Fig. 1. How many array accesses does a single call to the most expensive method of `X` take in the worst case?

A $\sim 4N$. B 2.
C $\sim 2N$. D 7.

(f) (*1 pt.*) Consider the class in Fig. 1. What is the amortized number of array accesses per operation in a sequence of k `insert` operations beginning in an empty data structure?

A linear in k. B constant.
C linearithmic in k. D quadratic in k.

(g) (*1 pt.*) Consider the class in Fig. 1. What is the amortized number of array accesses per operation in a sequence of k `remove` operations beginning in a data stucture with k elements?

A linear in k. B constant.
C linearithmic in k. D quadratic in k.

(h) (*3 pt.*) Consider the class in Fig. 1. What is the amortized number of array accesses per operation in a *worst case sequence* of k operations beginning in an empty data structure?
(Answer this question on a separate piece of paper. For full points, you need to specify the number of operations using tilde notation – if you can't do that, use asymptotic notation or "order of growth"-terminology. For full points, you need to describe which k operations are performed by a worst case sequence.)

```
public class X<Key extends Comparable<Key>>
{
    private Key[] A = (Key[]) new Comparable[1];
    private int N = 0;

    public void insert(Key in)
    {
        A[N] = in;
        N = N + 1;
        if (N == A.length) rebuild();
    }

    public Key remove() // assumes X is not empty
    {
        // search for maximum key:
        int max = 0;
        for (int i = 1; i < N; i++)
          if (A[i].compareTo(A[max]) > 0)
            max = i;
        // exchange with last element:
        Key tmp = A[max];
        A[max] = A[N-1];

        N = N - 1;
        return tmp;
    }

    private void rebuild()
    {
        Key[] tmp = (Key[]) new Comparable[2*A.length];
        for (int i = 0; i < N; i++ )
            tmp[i] = A[i];
        A = tmp;
    }
}
```

Figure 1: Class X.

Operation of common data structures

3.

(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 12 5 $*$ 3 7 $*$ $*$ $*$ 2 4 13 $*$ 14 15 $*$ $*$ $*$

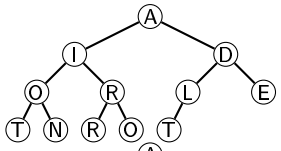
The output resulting from these operations is:

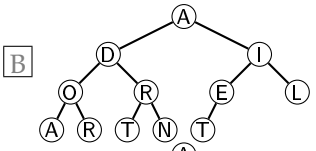
5 7 3 12 13 15 14 4

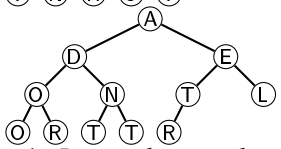
What is the data structure?

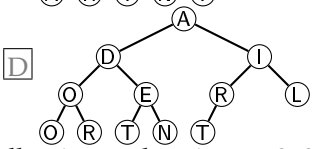
A A binary heap
B A multi-way heap
C A stack
D A FIFO queue

(b) (*1 pt.*) Assume keys I N T O T A L O R D E R are inserted in that order into an initially empty heap-based priority queue ([SW] p. 218f), using the `insert` method. What is the resulting data structure?

A 

B 

C 

D 

(c) (*1 pt.*) Insert the numbers 1 through 7 in the following orders into a 2–3 tree. Which ordering produces the tree of smallest height?

A 1 3 5 7 2 4 6
B 1 2 3 4 5 6 7
C 7 6 5 4 3 2 1
D 2 4 5 1 3 6 7

(d) (*1 pt.*) Which of the following is true about insertion sort, but not true about Quicksort?

A It uses only a constant amount of auxiliary space.
B It has linearithmic time complexity.
C It is in-place.
D It uses an optimal number of exchanges.

(e) (*1 pt.*) (CORRECTED) 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	10	0	12	3	7	4	5	6	8	1	0

We use the hash function `(key.hashCode() & 0x7fffffff) % 13`. 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 linear probing (p. 374). What is the resulting hash table?

A

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	A	O	N	Q	E	S	T	U	I		X		M
value	2	10	11	4	0	7	8	5	9		1		3

B

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	A	O		Q	E	E	S	U	T	I	X	N	M
value	2	10		4	6	0	7	5	8	9	1	11	3

Hint: $A < D < E < I < L < N < O < R < T$

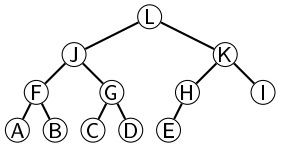
C

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	N	O	A	Q	E	E	S	U	T	I	X		M
value	11	10	2	4	0	6	7	5	8	9	1		3

D

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	A	O	N	Q	E	S	T	U	I		X		M
value	2	10	11	4	6	7	8	5	9		1		3

(f) (*1 pt.*) In which order could the nodes of this graph have been visited by a breadth first search starting in node A?



A A F B J G D L C K H E I

B A F B J G L C D K H I E

C A F B J G C D L K H E I

D A F B J G K L C D H I E

(g) (*1 pt.*) Give a trace for LSD string sort (sometimes called LSD radix sort) of 23 49 33 48 42.

A
```
23 42 23
49 23 33
33 33 42
48 48 48
42 49 49
```

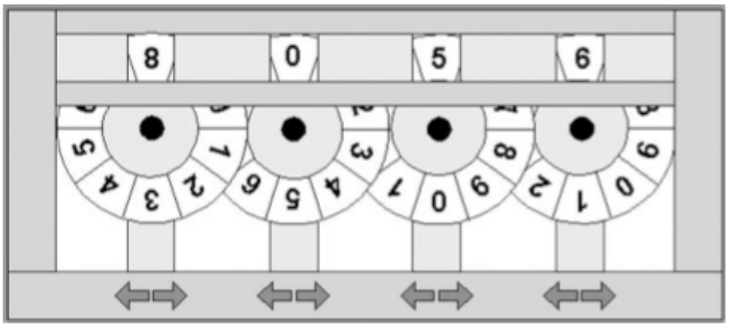
B
```
23 23 23
49 33 33
33 49 42
48 48 48
42 42 49
```

C
```
23 23 23
49 33 33
33 42 42
48 48 48
42 49 49
```

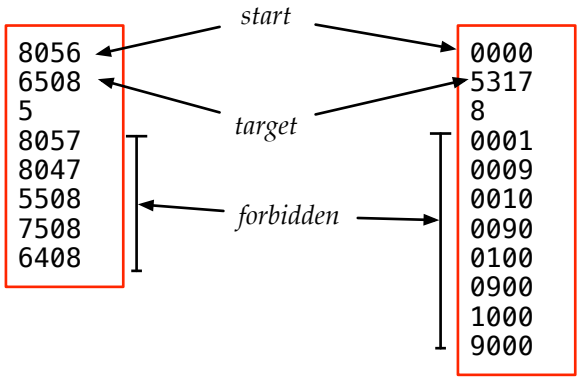
D
```
23 23 23
49 42 33
33 33 42
48 48 48
42 49 49
```

Design of algorithms

4. Consider a machine like this:



It consists of four wheels. Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the above figure the wheels form the integer 8056. Each wheel has two buttons associated with it. Pressing the button marked with a left arrow rotates the wheel one digit in the clockwise direction and pressing the one marked with the right arrow rotates it by one digit in the opposite direction.
The input consists of a start configuration, a target configuration, an integer k, and k forbidden configurations. Here are two small examples:



Typically, k could be a few thousand. Your job is to write a program to calculate the minimum number of button presses required to transform the initial configuration to the target configuration without passing through a forbidden configuration, or report that no solution is possible. For example, in the left example, the output should be "14", and in the right example it should be "impossible".
Answer these questions on a separate piece of paper:

(a) (*5 pt.*) Explain how you would solve this problem in an efficient way. You are encouraged to make use of existing algorithms, models, or data structures from the book. Estimate the running time of your construction. (Be short and precise. This question can be perfectly answered on half a page of text, maybe less. If you find yourself writing more than one page, you're using the wrong level of detail.)

(b) (*2 pt.*) Assume there are M wheels instead of 4. Express the growth rate of the running time of the algorithm in terms of M.