Answers of Algorithms and Data Structures, Exam 20 May 2019

The questions start on page 7.

It is strongly preferred that you fit your answers on these sheets. If you really must, you can use a separate sheet of paper instead. Please indicate that clearly.

Your name:

	1a	1b	1c	1d	1e	1f	1g	2a	2d	2e	2f	2g	3a	3b	3c	3d	3i	3j
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

MSc students receive no credit for 3j. BSc students receive no credit for 3k.

2b

2c

2h

2i

2j

3e

3f

3g

3h

3k

Min: Max:

MSc students receive no credit for 3j. BSc students receive no credit for 3k.

4a

4b

4c

Algorithms and Data Structures

Exam 20 May 2019

Thore Husfeldt and Riko Jacob, 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.) Electronic devices like mobile phones, pocket calculators, computers or e-book readers are not allowed.

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

Where to write. Mark you answers on pages 1–6. If you really have to, you may use separate sheets of paper for the free text questions instead, but please be clear about it (cross out everything and write "see separate paper, page 1" or something like that.) For the love of all that is Good and Holy, write legibly. Hand in pages 1–6, and any separate sheet(s) of paper. Do not hand in pages 7–15, they will not be read.

Exam questions

1. Analysis of algorithms

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

A $f(N) = 2N$ and $g(N) = N + \sqrt{N}$

B $f(N) = \sqrt{N} + \log N$ and $g(N) = \sqrt{N} + 2\log N$

C $f(N) = N \log N + N$ and $g(N) = N \log N + N^2$

D $f(N) = 2\sqrt{N} + N$ and $g(N) = \sqrt{N} + 2N$

(b) (*1 pt.*) Which pair of functions satisfy $f(N) = O(g(N))$?

A $f(N) = N$ and $g(N) = N/\log N$

B $f(N) = \sqrt{N} \cdot (\log N)$ and $g(N) = \sqrt{N}$

C $f(N) = (N+1) \cdot (N-1)$ and $g(N) = N^2$

D $f(N) = N^3$ and $g(N) = 4N^2 + 5N$

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

```
# python3
i = 1
while i < N:
    stdio.write("*")
    i = i*2
```

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

A $\sim \log_2 N$ B $\sim N/2$ C $\sim N$ D $\sim \frac{1}{2}N^2$

(d) (*1 pt.*) How many stars are printed? (Choose the smallest correct estimate.)

```
# python3
i = 0
while i < N:
    j = i
    while j > 0:
        j = j-2
        stdio.write("*")
    i = i+1
```

```
// java
for (int i = 0; i < N; i = i+1)
     for (int j = i; j > 0; j = j-2)
          StdOut.print("*");
```

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

(e) (*1 pt.*) What is the asymptotic running time of the following piece of code? (Choose the smallest correct estimate.)

```
# python3
i = 0
while i < N:
    i = i+1
    if i < 5:
        j = 0
        while j < N:
            A[i] = A[i] + A[j] + i
            j = j + 1
```

```
// java
for (int i = 0; i < N; i = i+1)
   if ( i < 5 )
      for (int j = 0; j < N; j = j+1)
          A[i] = A[i] + A[j] + i;
```

A linear in N B linearithmic in N C quadratic in N D cubic in N

(f) (*1 pt.*) Find a recurrence relation for the number of arithmitic operations (additions and subtractions) performed by the following recursive method. (Choose the smallest correct estimate. The base case is $T(0) = 0$ in all cases.)

```
# python3
def r( N ):
  if N > 0:
    return r(N-1) + r(N-1) + N
  else:
    return 2
```

```
// java
static int r(int N)
{
  if (N > 0)
    return r(N-1) + r(N-1) + N;
  else
    return 2;
}
```

A $T(N) = T(N-1) + 2$
B $T(N) = T(N-1) + T(N-1) + N$
C $T(N) = 2 \cdot T(N-1) + 4$
D $T(N) = T(N-1) + 4$

(g) (*1 pt.*) What is the asymptotic running time of the following piece of code? (Choose the smallest correct estimate.)

```
# python3
def f(x):
    return x + x

x = '*'
for i in range(N):
    x = f(x)
print(x)
```

```
// java
static String f( String x )
   { return x + x; }

String x = "*";
for (int i = 0; i < N; i = i + 1)
    x = f(x);
System.out.println(x);
```

A linear in N B linearithmic in N C quadratic in N D even slower

2. Class M. The next few questions all concern the class defined in Fig. 2 and 1.

(a) (*1 pt.*) Which well-known type of collection or mathematical structure does `M` implement with the provided methods?

A Graph B Set C Stack D Priority Queue

(b) (*1 pt.*) What is the result of the following operations?

```
insert(3)
contains(3)
contains(4)
insert(3)
reportmax()
insert(4)
contains(3)
```

Write a single line with the results of the operations that return a value.

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

```
insert(3)
contains(3)
contains(4)
insert(3)
reportmax()
insert(4)
contains(3)
```

(d) (*1 pt.*) What is the worst-case running time of `contains`?

A $O(\log N)$. B $O(N)$. C $O(N \log N)$. D $O(1)$.

(e) (*1 pt.*) What is the best-case running time of `contains`?

A $O(\log N)$. B $O(N)$. C $O(N \log N)$. D $O(1)$.

(f) (*1 pt.*) What is the amortised running time of `insert`?

A $O(\log N)$. B $O(N)$. C $O(N \log N)$. D $O(1)$.

(g) (*1 pt.*) How much space does the data structure use if I insert the same element K times?

A $O(\log K)$. B $O(K)$. C $O(K \log K)$. D $O(1)$.

(h) (*1 pt.*) Write a method `reportmin()` that returns the smallest inserted element. If the data structure is empty, you may return anything or your program may fail. Do not change the insert method.

(i) (*1 pt.*) Modify the data structure so that the space usage is proportional to the number of different items inserted. You may not change the worst-case running time of any of the methods. (But you may change their amortised running time.)

(j) (*1 pt.*) Explain how to implement a function `both(A,B)` that for two Objects `A` and `B` of class `M` returns a new object `C` of class `M` such that for any item `t` the call `C.contains(t)` returns true if and only if both calls `A.contains(t)` and `B.contains(t)` return true. Subsequent changes to `A` or `B` need not (and cannot) be reflected by `C`. The objects `A` or `B` may not be changed by `both`. Faster is better. Do not change any of the existing methods of class `M`.

```python
#python3

class M:
    def __init__(self):
        self.L = [None]
        self.N = 0
        self.max = None

    def insert(self, item):
        if len(self.L) == self.N:
            self.resize(2 * self.N)
        assert self.N < len(self.L)
        self.L[self.N] = item
        self.N += 1
        if self.max == None or  self.max < item:
            self.max=item

    def resize(self, newN):
        print("resize",newN)
        tmp = [None] * newN
        for i in range(self.N):
            tmp[i] = self.L[i]
        self.L = tmp

    def contains(self, item):
        for i in range(self.N):
            if self.L[i] == item:
                return True
        return False

    def reportmax(self):
        return self.max
```

Figure 1: Class `M` (for Mystery), python version.

```java
// java

public class M {
    int [] L = { 0 };
    int    N = 0;
    int  max = Integer.MIN_VALUE;

    void insert(int item) {
        if( L.length == N )
            resize(2*N);
        assert( N < L.length );
        L[N] = item;
        N += 1;
        if (item > max)
            max=item;
    }

    void resize(int newN) {
        int[] tmp = new int[newN];
        for(int i=0;i<N;i++)
            tmp[i] = L[i];
        L=tmp;
    }
    boolean contains(int item) {
        for(int i=0;i<N;i++)
            if( item == L[i] )
                return true;
        return false;
    }
    int reportmax() {
        return max;
    }
}
```

Figure 2: Class `M` (for Mystery), java version.

3. Operation of common algorithms and data structures.

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

5 3 9 $*$ 4

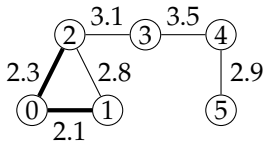
I now perform another `pop()`. What is returned?

A 3 B 4 C 5 D 9

(b) (*1 pt.*) Which comparison-based sorting algorithm sorts certain inputs in time $O(N)$?

A Selection sort B Insertion sort C Mergesort D Quicksort

(c) (*1 pt.*) Here is a graph in the middle of an execution of Kruskal's MST algorithm:



Edge weights are written next to the edges. The edges already added to the MST are drawn thick. Which edge is added to the MST next?

A 1-2 B 2-3 C 3-4 D 4-5

(d) (*1 pt.*) Let me sort `EXAM` using my favourite sorting method. The sequence of exchanges looks lik this:

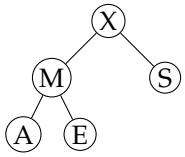
```
EXAM
AXEM
AEXM
AEMX
```

Which sorting algorithm am I using?

A Selection sort B Insertion sort C Mergesort D Quicksort

(e) (*1 pt.*) Insert the 4 letters `MEXA` (in that order) into a binary search tree [SW 3.2] using lexicographic ordering. Draw the result.[1]

(f) (*1 pt.*) Here's a heap:



Draw the heap after a single call to `delMax`.

[1] Recall the lexicographic order of the English alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

(g) (*1 pt.*) Use hashing with separate chaining [SW 3.4] to process the following list of insertions of key–value pairs from left to right:

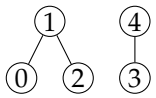
key	M	E	X	I	C	O	E	X	A	M
value	0	1	2	3	4	5	6	7	8	9

(So the first key is `M`, with value 0.) The hash code of a letter is its ASCII value modulo m, where $m = 4$; to spare you the calculations, here are the relevant hash codes:

key	A	C	E	I	M	O	X
hash code	1	3	1	1	1	3	0

Draw the resulting data structure.

(h) (*1 pt.*) I'm using Weighted Quick-Union. After some operations, my data structure looks like this:



I now perform `union(3,0)`. Draw the result.

(i) (*1 pt.*) I have a sequence A of N integers between 1 and M, where $N \leq 10^8$ and $M \leq 10^8$. I want to make sure that A does not contain the same value three times. What's the *worst* way of doing this?

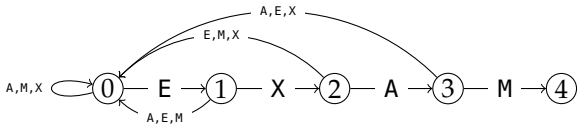
A Three nested loops of indices i, j, and k, to find $0 \leq i < j < k < N$ with $A[i] = A[j] = A[k]$.

B Sort A, then loop through the sorted sequence once to detect three equal entries in a row.

C A symbol table based on a red–black search tree to implement a frequency counter.

D Hashing with separate chaining using a hash function with a table size of roughly M.

(j) (*1 pt.*) **BSc only.** Here is the DFA for the pattern `EXAM` over the alphabet `AEMX`:



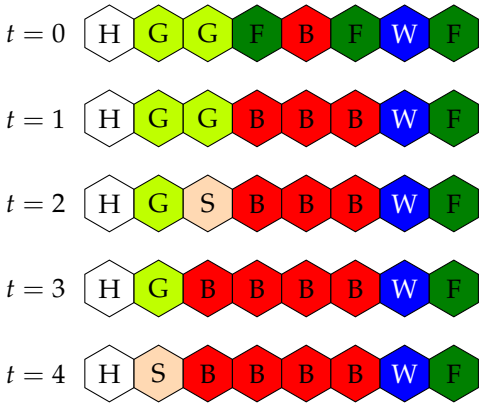
Which state is reached on input text `EXEMAE`?

A 0 B 1 C 2 D 3

(k) (*1 pt.*) **MSc only.** Consider a B-Tree with parameter $M = 1001$. What is the minimal and maximal number of items that can be stored in such a B-tree of height precisely 3?

4. Design of algorithms

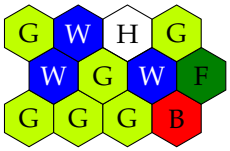
The world consists of R rows and C columns of six-sided tiles. A tile is either F (forest), G (grass), B (burning), S (smouldering[2]), W (water), or H (house). Time proceeds in discrete steps. Fire spreads using the following rules: Grass tiles next to a burning tile become smouldering in one step. A smouldering tile becomes burning in one step. Forest and house tiles next to a burning tile become burning in one step. Water tiles never change. Here is an example, with $R = 1$ and $C = 8$:



In the above example, the house is on fire at time $t = 6$.

(a) *(2 Pt.)* Assume $R = 1$. Give a detailed description, preferably using pseudocode, of an algorithm that computes the time t when the H-tile catches fire. You can assume that the input is given as a sequence $T[0], \ldots, T[C-1]$ of characters from `HGFBW`; for instance $T[1] =$ `G` in the above example. There is a single `H` and a single `B`. State the running time in terms of R and C.

(b) *(2 Pt.)* (Now we no longer assume $R = 1$.) Describe an algorithm that computes the time t when the H-tile catches fire. For instance, the input could look like this:



You can assume that the input is given as a two-dimensional array $T[r][c]$ of characters from `HGFBW`; with $0 \le r < R$ and $0 \le c < C$. Let's agree that $T[1][0]$ is the south-east neighbour of $T[0][0]$. In the above example, $T[1][2] =$ `W`. There is a single `H` and a single `B`. You should model the problem as a graph problem. Your explanation *must* include a careful and complete drawing of the graph corresponding to the above example input. It must be clear if the graph is undirected or directed, and which weights (if any) are on the edges. In general, how many vertices and how many edges does the graph have in terms of R and C? State the running time of your algorithm in terms of R and C.

(c) *(2 Pt.)* Assume the top-left tile is always an island, i.e., $T[0][0] =$ `G` and $T[0][1] = T[1][0] =$ `W`, just like in the previous example. Alice starts in the house, and wants to get to $T[0][0]$, where she'd be safe from the fire. Describe an algorithm that determines if Alice can make it. Alice moves exactly one tile per time step (no matter the terrain). She dies if she is on a burning or smouldering tile. To make the question well-defined, let's agree that she 'moves first' in the sense that she can successfully leave a tile at the same time step that it catches fire.

In all questions, use existing algorithms and data structures as much as you can, write clearly, and be brief.

[2] smoulder (verb): to burn slowly with smoke but no flame: the bonfire still smouldered, the smoke drifting over the paddock. In Danish, *ulme*.