Answers of Algorithms and Data Structures, Exam 20 August 2019

The questions start on page 5.

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.

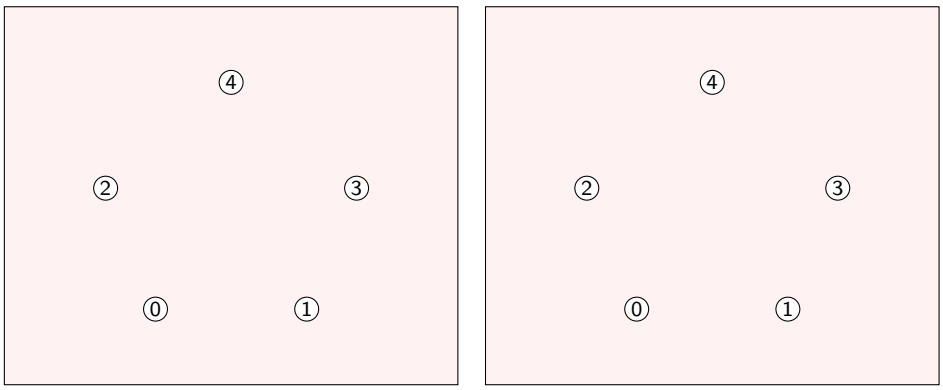
Your name:

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

2e

2h

3d



3g

3h

3i

4a

4b

Algorithms and Data Structures

Exam 20 August 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. 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. (A question that is not multiple-choice cannot give you negative points.)

Where to write. Mark you answers on pages 1–4. 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 1–4, and any separate sheet(s) of paper. Do not hand in pages 5–11, they will not be read.

Exam questions

1. Analysis of algorithms

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

A $f(N) = 3N$ and $g(N) = N + \sqrt{N}$

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

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

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

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

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

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

C $f(N) = (N+1) \cdot (N-1)$ and $g(N) = N^{3/2}$

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 + 3
```

```
// java
for (int i = 1 ; i < N; i = i + 3)
    StdOut.print("*");
```

A $\sim \log_3 N$

B $\sim N/3$

C $\sim 3N$

D $\sim \frac{1}{3}N^3$

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

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

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

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

(e) (*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) + 4
  else:
    return 2
```

```
// java
static int r(int N)
{
  if (N > 0)
    return r(N-1) + 4;
  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 $T(N) = T(N-1) + 4$

(f) (*1 pt.*) What is the asymptotic running time of the following piece of code? (Choose the smallest correct estimate. Recall that in both Python and Java, string concatenation takes time linear in the length of the resulting string.)

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

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

```
// java
static String f( String x )
   { return 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

```python
def is_square(x):
    ''' return True if x is a square number '''
    i = 0
    while True:
        if i * i == x:
            return True
        if i * i > x:
            return False
        i += 1

def f(L):
    ct = 0
    for x in L:
        if is_square(x):
            ct += 1
    return ct
```

Figure 1: Function `f`, python version.

```java
    static boolean is_square(double x)
    { // return true if x is a square number
        int i = 0;
        while (true) {
            if (i * i == x)
                return true;
            if (i * i > x)
                return false;
            i += 1;
        }
    }

    static int f(List<Double> L)
    {
        int ct = 0;
        for (double x: L)
            if (is_square(x))
                ct = ct + 1;
        return ct;
    }
```

Figure 2: Function `f`, java version.

2. Function f. The next few questions all concern the function defined in Fig. 1 and 2. Unless otherwise noted, L is a list of nonnegative integers of length N. Recall that a *square* (also called *square number*, *kvadrattal* in Danish; 9 is an example, 10 is not) is a number x such that there exists an integer i with $i^2 = x$. Let's agree to treat $0^2 = 0$ a square.

(a) (*1 pt.*) What does `f` do?

- [A] Returns the number of squares in a given list of positive integers.
- [B] Decides if a given number is a square.
- [C] Computes the square of the number of elements in a list.
- [D] Counts the distance to the nearest square of a given number.

(b) (*1 pt.*) How many calls to `is_square` are performed in the *worst* case?

[A] $O(\log N)$. [B] $O(N)$. [C] $O(N \log N)$. [D] $O(N^{3/2})$.

(c) (*1 pt.*) How many calls to `is_square` are performed in the *best* case?

[A] $O(\log N)$. [B] $O(N)$. [C] $O(N \log N)$. [D] $O(N^{3/2})$.

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

[A] $O(\log N)$. [B] $O(N)$. [C] $O(N \log N)$. [D] $O(N^{3/2})$.

(e) (*1 pt.*) Write a function `g` that takes a list of nonnegative integers and returns the number of *non-squares*. (You can refer to the declarations in Fig. 1 or 2 in your code.)

(f) (*1 pt.*) What happens if I run `f` on a list not consisting entirely of positive integers, such as $L_1 = [4, -3]$ or $L_2 = [4.0, 3.141]$?

- [A] The running time becomes quadratic.
- [B] It gives the right answer on L_2 but doesn't even terminate on L_1.
- [C] It gives a wrong answer in both cases.
- [D] It is still correct.

(g) (*1 pt.*) Professor Eager wants to improve the performance of `is_square`. What is the *worst* of his suggestions?

- [A] Use binary search to get the time down to $O(\log x)$.
- [B] Use the square root function from the math library to get down to constant time.
- [C] Rewrite the `while`-loop as a `for`-loop.
- [D] Increase `i` by 2 (rather than 1) i line 9 , making the code twice as fast.

(h) (*1 pt.*) Explain how to change `f` so that is counts the *different* squares in the given list. For instance, if the list is $[4, 3, 9, 4]$ then the answer is 2, because the two squares 2^2 and 3^2 appear in the list. You are welcome to use pseudocode, but precise prose is also fine. State the running time. Faster is better.

3. Operation of common algorithms and data structures.

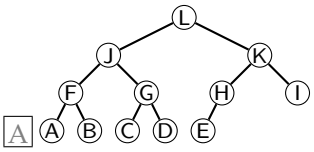
(a) (*1 pt.*) Consider the following sequence of operations on a Stack, where a number i means `push(`i`)` and "$*$" means `pop()`. The data structure is initially empty.

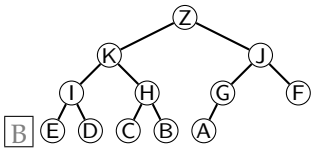
5 3 1 $*$ 8

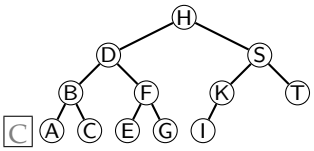
I now perform another `pop()`. What is returned?

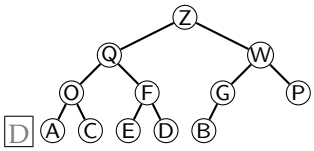
A 3 B 8 C 5 D 1

(b) (*1 pt.*) Which of the following trees is a search tree?

A 

B 

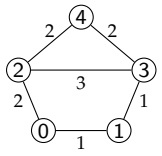
C 

D 

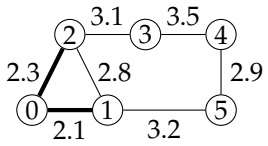
(c) (*1 pt.*) Which algorithm sorts its inputs in time $O(N \log N)$ in the worst case?

A Selection sort B Insertion sort C Mergesort D Quicksort

(d) (*1 pt.*) Draw two different minimum spanning tree for



(e) (*1 pt.*) Here is an undirected graph in the middle of an execution of Dijkstra's algorithm, computing a shortest-path tree from vertex 0.



Edge weights are written next to the edges, all edges are drawn undirected. The edges already added to the shortest-path tree are drawn thick. Which edge is added to the shortest-path tree next?

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

(f) (*1 pt.*) Let me sort `DCBA` using my favourite method. The sequence of exchanges begins like this:

```
DCBA
CDBA
CDAB
```

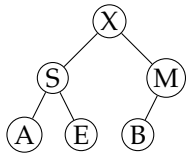
Which sorting algorithm am I using?

A Selection sort B Insertion sort C Mergesort D Quicksort

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

[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.

(h) (*1 pt.*) Here's a heap implemening a priority queue:



Draw the heap after two calls to `delMax`.

(i) (*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	B	A	N	A	N	A	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	B	E	M	N	X
hash code	1	2	1	1	2	0

Draw the resulting data structure.

(j) (*1 pt.*) In the resizing-array implementation of a pushdown (LIFO) stack, the `pop` method calls `resize` if the number N of stack elements equals $\frac{1}{4}$ of the length of the underlying array `a[]`. What happens when I change that value to $\frac{1}{2}$? (In words, "when I halve the array when it becomes half full?")

A The amortised time per stack operation becomes linear.

B A runtime exception.

C The `push` and `pop` methods are no longer correct.

D You waste space.

(k) (*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 check if A contains three identical consecutive values. For instance, the sequence $[1, 6, 2, 2, 2, 5, 5, 2]$ does, but the sequence $[1, 1, 2, 2, 3, 2]$ does not. What's the *best* way of doing this?

A A single loop with an index i running from 0 to $N - 2$ to find $A[i] = A[i + 1] = A[i + 2]$.

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

C Three nested loops with indices i, j, and k, each running from 0 to $N - 1$ to find $A[i] = A[j] = A[k]$ and such that $i + j + k = 3i + 3$.

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

4. Design of algorithms

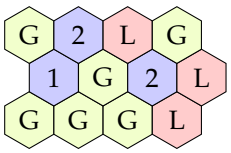
The world consists of R rows and C columns of six-sided tiles. A tile is either grass (G), lava (L) or a certain depth of water (given as a positive integer between 1 and RC, in centimeters, of depth below the surface). Here are some examples, with $R = 1$ and $C = 8$:

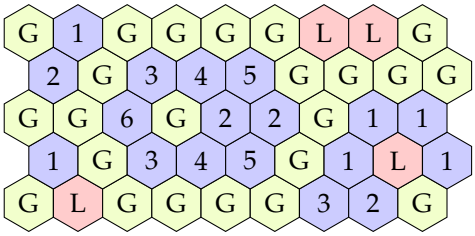
Your goal is to lower the water level of the world (as little as possible), turning water tile into grass tiles, so that all grass tiles become connected and the inhabitants in the world can visit each other. More precisely, every pair of G-tiles must be connected by a sequence of neighbouring G-tiles. (You can pretend that you are the chief engineer of terraforming on a new interplanetary space colonialisation mission. Or you are some kind of god. Or you are a climate scientist making models. Or it's a video game. Whatever.) It takes one day to decrease the water level in the world by one centimeter, equivalently, to decrease the depth of every water tile by 1. When a water tile is at depth 0, it becomes a grass tile. Lava tiles are unaffected by the water level. In the example to the right, the goal can be achieved in three days. In the example to the left, the goal is `impossible` to achieve.

The travel time for the inhabitants is not important. (You can imagine that they are infinitely fast and in infinitely good shape, except unable to swim. Or maybe they have access to these very fast electric scooters you see everywhere now.)

(a) *(3 Pt.)* Assume $R = 1$. Give a detailed description, preferably using pseudocode, of an algorithm that computes the minimum time (in number of days, as an integer) before all grass tiles are connected, or writes `impossible`. You can assume that the input is given as a one-dimensional array $T[c]$ of integers; with $0 \leq c < C$, and where 0 means 'G' and -1 means 'L'.

(b) *(4 Pt.)* Same question as above, but now we no longer assume $R = 1$. Here are two example worlds, both with answer "two days":





You can assume that the input is given as a two-dimensional array $T[r][c]$ of integers; with $0 \leq r < R$ and $0 \leq c < C$, and where 0 means 'G' and -1 means 'L'. Let's agree that $T[0][0]$ descrbes the north–west-most tile, $T[0][1]$ its neighbour to the east, and $T[1][0]$ its neighbour to the south–east. For instance, the example world to the left is encoded as $[[0, 2, -1, 0], [1, 0, 2, -1], [0, 0, 0, -1]$.

Maybe you want to model the problem as a graph problem. In that case, your explanation *must* include a careful and complete drawing of the final graph corresponding to the example world to the left (the smaller one). 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 (asymptotically) does the graph maximally have, in terms of R and C?

Describe your algorithm, preferably in precise prose rather than pseudocode. State the total running time of your algorithm in terms of R and C.

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