Answers of Algorithms and Data Structures, Exam 28 May 2018

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	1g	2a	2d	2e	2f	3a	3b	3d	3f	3h	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 3i.

2b

2c

2g

Base revision 5060ff3. . ., Fri May 17 23:11:04 2019 +0200, Riko Jacob @X260.

2h

2i

3c

3e

3g

4a

4b

Algorithms and Data Structures

Exam 28 May 2018

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.)

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 1–4. 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 5–13, 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 $f(N) = N$ and $g(N) = N + N^2$

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

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

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

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

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

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

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

D $f(N) = N^3$ and $g(N) = 3N^2 + 8N$

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

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

```
// 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:
    i = i+1
    j = i
    while j > 0:
        j = j//2
        stdio.write("*")
```

```
// 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
k = 0
while i < N:
    i = i+1
    if k > 0:
        j = 0
        while j < N:
            A[i] = A[i] + A[j] + k
            j = j + 1
    k = 1 - k;
```

```
// java
k = 0;
for (int i = 0; i < N; i = i+1) {
   if ( k > 0 )
      for (int j = 0; j < N; j = j+1)
          A[i] = A[i] + A[j] + k;
   k = 1 - k;
}
```

[A] linear in N [B] linearithmic in N [C] quadratic in N [D] cubic 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) + 2 + N
    else:
       return 2
```

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

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

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

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

D $T(N) = T(N) + 2 + N$

(g) (*1 pt.*) Assume I have a function `f(int K)` that runs in amortised constant time in K, but logarithmic worst case time. What is the running time of

```
# python3
for i in range(N):
    f(i)
```

```
// java
for (int i = 0; i<N; i=i+1)
    f(N);
```

(Choose the smallest correct estimate.)

A Linearithmic in N.

B Linear in N.

C Quadratic in N.

D Impossible to say from the information given.

```java
import edu.princeton.cs.algs4.*;

public class S
{
    public int[] a;
    int sz;

    public S(int cap)
    { a = new int[cap]; sz = 0;}

    public void push(int item)
    {
        if (sz == a.length) shift();
        a[sz] = item;
        sz++;
    }
    public int pop()
    {
        sz--;
        return a[sz];
    }

    private void shift()
    {
        int[] tmp = new int[a.length];
        for (int i = 0; i<a.length - 1; i++) tmp[i] = a[i+1];
        a = tmp;
        sz--;
    }
}
```

Figure 1: Class S (for Strange Stack), java version.

```
class S:
    def __init__(self, capacity):
        self.a = [0] * capacity;
        self.sz = 0

    def push(self, item):
        if self.sz == len(self.a):
            self.shift()
        self.a[self.sz] = item
        self.sz += 1

    def pop(self):
        self.sz -= 1
        return self.a[self.sz]

    def shift(self):
        tmp = [0] * len(self.a)
        for i in range(0, len(self.a) - 1):
            tmp[i] = self.a[i+1]
        self.a = tmp
        self.sz -= 1
```

Figure 2: Class `S` (for Strange Stack), python version.

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

(a) (*1 pt.*) Class *S* is some kind of fixed-capacity stack-like data structure. When an object of class *S* runs out of capacity, it

- A forgets the oldest element.
- B ignores the next push.
- C resizes itself so as to make room for more elements.
- D return the smallest element.

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

```
// java
int N = 4;
S s = new S(N);
for (int i = 0; i < N    ; i++)
    s.push(i);
for (int i = 0; i < N    ; i++)
    StdOut.print(s.pop());
for (int i = 0; i < N + 1; i++)
    s.push(i);
for (int i = 0; i < N    ; i++)
    StdOut.print(s.pop());
```

```
# python3
N = 4;
s = S(N)
for i in range(N):
    s.push(i)
for i in range(N):
    stdio.write(s.pop())
for i in range(N + 1):
    s.push(i)
for i in range(N):
    stdio.write(s.pop())
```

(Ignore whitespace such as blanks or newlines.)

(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.)

```
// java                    # python 3
S s = new S(3);            s = S(3)
s.push(5);                 s.push(5)
s.push(6);                 s.push(6)
s.pop();                   s.pop()
```

(d) (*1 pt.*) Assume I initialised an S-object with capacity K. Now I perform a sequence of N pushes. What is the worst-case (also the worst possible dependency of K on N) running time of a single `push`? (Choose the smallest correct estimate.)

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

(e) (*1 pt.*) Assume I initialised an S-object with capacity K. Now I perform a sequence of N pushes, followed by N pops. What is the worst-case (also the worst possible dependency of K on N) running time of a single `pop`? (Choose the smallest correct estimate.)

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

(f) (*1 pt.*) Assume I initialised an S-object with capacity K. Now I perform N pushes and $N + 1$ pops, where $N \leq K$. What happens? **The java and the python version behave differently; taken out**

A At least one of the pushes or pops takes linear time.

B The program aborts with a runtime error or exception.

C The underlying array resizes to $\frac{1}{4}$ of its size.

D The smallest element is returned.

(g) (*1 pt.*) Add a method with the following signature that checks if the data structure is empty.

```
// java                          # python 3
public boolean isEmpty()         def is_empty(self):
```

Don't change any other methods in class `S` *and don't introduce new instance variables to* `S`.

(h) (*1 pt.*) Extend class `S` with the following signature that removes the element that was first pushed (i.e., the 'oldest' element). The method returns nothing. *Don't change any other methods in class* `S` *and don't introduce new instance variables to* `S`.

```
// java                          # python 3
public void remove_oldest()      def remove_oldest(self):
```

(i) (*2 pt.*) Explain how to change the implementation of the data structure so that push and pop take constant time in the worst case. The functionality must be unchanged. If K is the capacity, then the constructor should take $O(K)$ time and the data structure should take $O(K)$ space.

3. Operation of common algorithms and data structures.

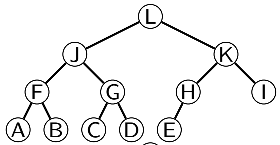
(a) (*1 pt.*) Consider the following sequence of operations on a data structure, where a number i means `insert(`i`)` and "$*$" means `remove()`. The data structure is initially empty.

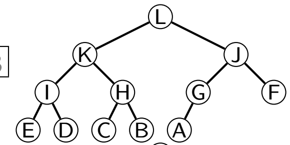
$$5 \quad 3 \quad 9 \quad *$$

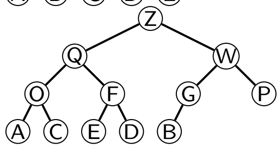
What is the data structure if the removed elements is 5 ?

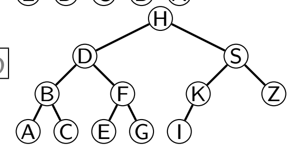
A Priority queue B Stack C Queue D Trie

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

A 

B 

C 

D 

(c) (*1 pt.*) Insert the keys `2 1 3 4 5` in that order into a binary search tree (without any rebalancing, using algorithm 3.3 in [SW]). Draw the result.

(d) (*1 pt.*) Assume I sort the letters `D C B A` into alphabetically increasing order using an unknown sorting algorithm. At some time during the process, the letters are arranged like this: `C D A B`. Which algorithm could I be using.

A Mergesort. B Quicksort. C Insertion sort. D Selection sort.

(e) (*1 pt.*) In the style of the book, draw the trie for the following key–value pairs:

key	value
ABLE	4
ABEL	3
ABE	2
BE	1
E	0

(f) (*1 pt.*) Consider a connected undirected graph with nonnegative integer weights. Let M be a minimum spanning tree in the graph. Let S be a shortest-paths tree in the graph. Let $w(M)$ and $w(S)$ denote the total weight of M and S, respetively. Which claim is always true?

A $w(M) \leq w(S)$. B $w(S) < w(M)$. C $w(S) = w(M)$. D $w(S) \neq w(M)$.

(g) (*1 pt.*) Insert the letters `D B A C` into a red-black BST in that order. Draw the resulting structure in the style of the book, use a fat edge to represent red links. (Your answer will be photocopied in black and white, so don't use fancy colours.)

(h) (*1 pt.*) Let's sort 4 strings using LSD string sort (algorithm 5.1 in [SW], with $N = 4$, $W = 3$). Here's a partial trace in the style of the book:

input	$d = 2$	$d = 1$	$d = 0$
`HAT`	`ADA`	...	
`CAT`	...	...	
`BOB`	...	$\boxed{\quad\quad}$	
`ADA`	...	...	

Which string goes into the box?

A `ADA`. B `BOB`. C `CAT`. D `HAT`.

(i) (*1 pt.*) **MSc only.** Professor Precipitat wants to sort n integers from the range $\{1, \ldots, 2n\}$. Reading his old course book, he has found a method for small integers called key-indexed counting (Sec. 5.1 in Sedgewick–Wayne), which runs in time $11n + 8n + 1$ according to Proposition A (Sec. 5.1). This confuses him, because he also remembers a lower bound of $n \log n$ on the complexity of sorting (Proposition I, Sec. 2.2). Help the good professor out by explaining why the two claims aren't in contradiction.

A Key-indexed counting is not a compare-based algorithm.

B $11n + 8n + 1 \sim n \log n$ for large n

C The worst-case performance of key-indexed counting can be much worse.

D Key-indexed counting is not optimal with respect to space usage.

(j) (*1 pt.*) **BSc only.** Consider the two sorting algorithms heapsort and quicksort without initial randomization of the input. In an application setting, we observe experimentally that for a wide range of input sizes, in particular for large inputs, the number of accesses to the arrays of quicksort is five times as large as for heapsort. Still, the running time of quicksort is 10 percent faster than heapsort. Which of the following explanations is reasonable

A Heapsort performs many more comparisons than Quicksort.

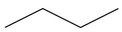
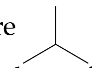
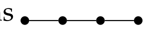
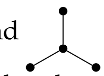
B Quicksort performs many more comparisons than Heapsort.

C Quicksort is asymptotically faster then Heapsort.

D The memory access pattern of Heapsort is much less cache friendly

4. Design of algorithms

In chemical graph theory, the *Wiener index* (also called *Wiener number*, and the *path number* by Wiener himself) is a number associated with a molecule and defined as follows.

For our purposes, a molecule is given by its so-called *molecular graph*: this is an undirected graph whose vertices are the non-hydrogen atoms and whose edges are the bonds. For instance, the organic compound Butane (C_4H_{10}) has two different molecular graphs (called 'structural isomers'): n-butane is the 4-vertex path  and Isobutane is the branched structure  . (Chemists don't traditionally draw the atoms at the endpoints of the bonds. Non-chemists might prefer to imagine fat dots at the endpoints of each line, such as  and  .)

The Wiener index is now defined as the sum of the distances between each pair of vertices in the molecular graph. (Apparently this has to do with the boiling point of the compound. Not important for us.) Recall that the distance between a pair of vertices is the length of a shortest path between them. For our two example compounds, their Wiener indices are

$$\text{n-butane: } 1+1+1+2+2+3 = 10$$

and

$$\text{Isobutane: } 1+1+1+2+2+2 = 9\,.$$

(These examples show that every pair of vertices contributes *once*.)

A chemical compound is given as follows. If there are B bonds then the input consists of $B+2$ lines. The first two lines contain A, the number of atoms (which we enumerate $0, 1, \ldots, A-1$, followed by B, the number of bonds. The following lines contains a pair of numbers each, describing the endpoints of each bond. For instance, here are n-butane (left) and Isobutane (right):

```
4           4
3           3
0 1         0 1
1 2         0 2
2 3         0 3
```

(a) (*4 pt.*) Describe an algorithm that computes the Wiener index of a given chemical compound.

(b) (*2 pt.*) (Very difficult.) Assume the chemical graph is a *tree*, as in the Butane examples. Describe a linear-time algorithm for the problem.
For partial credit, solve the problem for the special case where the graph is a nicely balanced binary tree. (This is still difficult.)

For both questions, state the running time of your resulting algorithms in terms of the given parameters (such as A and B). You are strongly 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). Be short and precise. Each 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. You don't need to 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 solutions, as well as clarity of explanation.

Don't worry. Absolutely no understanding of chemistry is needed for this exercise.