Answers of Algorithms and Data Structures, Exam 24 May 2023

The questions start on page 1.

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:

Algorithms and Data Structures

Exam 24 May 2023

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 your answers on pages 1–**??**. 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–**??**, and any separate sheet(s) of paper. Do not hand in pages 1–10, they will not be read.

Exam questions

1. Analysis of algorithms

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

A $f(N) = 2^N$ and $g(N) = 2^{N+1}$

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

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

D $f(N) = \sqrt{N} + N$ and $g(N) = \sqrt[3]{N} + N$

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

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

B $f(N) = N \cdot N$ and $g(N) = N + N$

C $f(N) = (N + 1) \cdot \sqrt{N}$ and $g(N) = (N - 1) \cdot (N - 1)$

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

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

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

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

A $\sim \log_2 N$ B $\sim 2\log_2 N$ C $\sim N$ D $\sim 2N$

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

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

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

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

(e) (*1 pt.*) Consider the linked list data structure [SW 1.3]. What would be a good cost model ([SW 1.4]) for analysing the running time of insertions and deletions? sol: C

A Array accesses

B Arithmetic operations (additions, subtractions, multiplications)

C Variable assignments

D Space

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

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

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

A $T(N) = T(N-1) + 4$

B $T(N) = T(N-1) + T(N-2) + 4$

C $T(N) = T(N-1) \cdot T(N-2) + N$

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

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

```
# python3
a = A()
for i in range(N):
    a.f(N)
```

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

(Choose the smallest correct estimate.) sol: A

- A Linearithmic in N.
- B Amortised linear in N.
- C Linear in N.
- D Quadratic in N.

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

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

x = "Hello World!"
for i in range(N):
    x = f(x)[:5]
print(x)
```

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

String x = "Hello World!";
for (int i = 0; i < N; i = i + 1)
    x = f(x).substring(0,5);
System.out.println(x);
```

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

Remember that in both languages, string concatenation takes time proportional to the length of the resulting string.

```python
class V:
    def __init__(self, m: int):
        self.V = [0] * m
        self.n = 0

    def insert(self, i: int):
        self.n += 1
        self.V[i] += 1

    def size(self) -> int:
        return self.n

    def remove(self) -> int:
        # assumes collection is not empty
        self.n -= 1
        i = 0
        while i < len(self.V):
            if self.V[i] != 0:
                self.V[i] -= 1
                return i
            i += 1
        return -1  # This shouldn't happen
```

Figure 1: Class `V`, python version.

```java
class V {
    int[] V;
    int n;

    public V(int m) {
        V = new int[m]; // by java convention: V = [0,...,0]
        n = 0;
    }

    public void insert(int i) {
        n += 1;
        V[i] += 1;
    }

    public int size() { return n; }

    public int remove() {
        // assumes collection is not empty
        n -= 1;
        int i = 0;
        while (i < V.length) {
            if (V[i] != 0) {
                V[i] -= 1;
                return i;
            }
            i += 1;
        }
        return -1; // This shouldn't happen
    }
}
```

Figure 2: Class V, java version.

2. Class V. The next few questions all concern the class defined in Fig. 1 and 2. We use n and m to refer to the value of the instance variable and constructor parameter of the same name.

(a) (*1 pt.*) What is printed? (Write a single line, use spaces instead of line breaks) sol: 3

```
V v = new V(10);
v.insert(8);
v.insert(9);
v.insert(8);
v.insert(5);
System.out.println(v.remove());
System.out.println(v.remove());
System.out.println(v.remove());
```

```
v = V(10)
v.insert(8)
v.insert(9)
v.insert(8)
v.insert(5)
print(v.remove())
print(v.remove())
print(v.remove())
```

(b) (*1 pt.*) Clearly, class `V` implements the core functionality of some kind of collections data type. Which one? sol: A

- A Min-oriented priority queue (“`MinPQ`”) over $\{0, \dots, m-1\}$
- B A graph with m edges.
- C A double-ended queue of m integers
- D A set of m elements supporting membership queries

(c) (*1 pt.*) Draw the data structure after the following operations: sol: +16

```
# python3
v = V(5)
v.insert(2)
v.insert(3)
v.insert(2)
```

```
// java
V v = new V(5);
v.insert(2);
v.insert(3);
v.insert(2);
```

(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, including all instance variables.)

(d) (*1 pt.*) Assume $m > 1$. What is the worst-case running time of `insert`? (Choose the smallest correct estimate.) sol: D

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

(e) (*1 pt.*) Assume $m > 1$. What is the worst-case running time of `remove`? (Choose the smallest correct estimate.) sol: B

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

(f) (*1 pt.*) Assume $m > 1$. What is the best(!)-case running time of `remove`? (Choose the smallest correct estimate.) sol: D

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

(g) (*1 pt.*) Write a method `different()` that returns the number of *different* elements in the collection. Give the running time in terms of n and/or m. Pseudocode is fine; using existing library functions or data structures is fine; but you cannot change other parts of the data structure. sol: +20

(h) (*1 pt.*) Describe how the `different` method from the previous question could be supported in worst-case constant time. (This requires changing other methods.) sol: +23

(i) (*1 pt.*) How much space does the data structure use after n elements were inserted into an empty collection, k of which are different? sol: A

A $O(m)$ B $O(k)$ C $O(n)$ D $O(\max\{n, m, k\})$

(j) (*1 pt.*) Give a sequence of operations that shows that the running time for *remove* is not $O(n)$.
sol: 13

(k) (*1 pt.*) Bob would like to speed up *remove*, and suggests to store in an instance variable the value of the smallest element in the collection. This is easy to maintain under insertions with a single extra comparison. Explain to Bob why this will not improve the worst-case running time of *remove*. sol: A

A Remove is still slow, because when a small element is removed, and all the other elements are large, it takes linear time in m to find the 'new' smallest element.

B The *amortised* time for remove improves to $O(\log n)$ and the worst-case time is therefore n times that, *i.e.*, $O(n \log n)$, which is worse if n is of the same order of magnitue as m.

C This only works if every element in the collection is unique. (In other words, if $V[i] \in \{0, 1\}$ for all i.)

D This would work if you also rebuilt the data structure to use an array of twice the length (i.e., $2m$) every time some element i appears m times, much like for `Stack`.

3. Operation of common algorithms and data structures.

(a) (*1 pt.*) Consider the following sequence of operations on Double-ended queue (sometimes called *deque*). The data structure is initially empty.

`pushLeft(1)` `pushRight(2)` `pushRight(3)` `popLeft()` `pushLeft(4)` `popRight()` `popRight()`

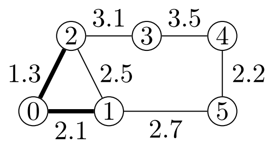
I now perform antoher `popRight`. What is returned? sol: C

A 2 B 3 C 4 D an error

(b) (*1 pt.*) Which comparison-based sorting algorithm sorts N inputs using $\sim \frac{1}{2}N^2$ comparisons in the *best* case? sol: A

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 will be the *last* edge that is added to the MST? sol: B

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

(d) (*1 pt.*) I'm running selection sort on `M I N K`.[1] What is the situation after the first exchange?
sol: 3

(e) (*1 pt.*) Insert the 4 letters `MINK` (in that order) into a left leaning red-black binary search tree [SW 3.3] using alphabetic ordering. Draw the result. sol: 9

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

key	M	E	T	T	E
value	0	1	2	3	4

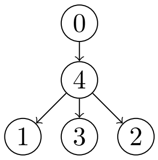
We use the hash function `(key.hashCode() & 0x7fffffff) % 7`. To spare you the calculations, the hash values are here:

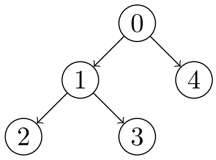
key	F, M, T	G, N, U	A, H, O, V	B, I, P, W	C, J, Q, X	D, K, R, Y	E, L, S, Z
hash	0	1	2	3	4	5	6

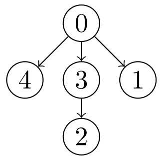
The key-value pairs are inserted from left to right into an initially empty hash table of size 7 using separate chaining. Draw the result. sol: +14

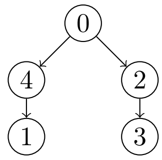
[1] Recall the alphabetic order of the English alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

(g) (*1 pt.*) I've performed breadth-first-search on a graph G and visited the nodes in the following order: 0, 4, 1, 3, 2. Which of the following graph is guaranteed to *not* be G? sol: D

A 

B 

C 

D 

(h) (*1 pt.*) I've invented a new sorting algorithm called shufflesort. Here it is in pseudocode, assuming a is an array with $n \geq 2$ elements of some comparable type. sol: A

> **repeat**
> $a = \mathit{shuffle}(a)$
> **until** $a_1 \leq \cdots \leq a_n$

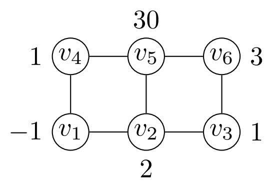
For *shuffle* I'll use the linear-time Knuth shuffle also used in Quicksort; the condtition after **until** can be checked in linear time using a linear scan. What is true about the running time of shufflesort?

- A Even slower than expected exponential time $O(2^n)$.
- B Linear time $O(n)$, when a was already sorted from the start.
- C Worst case cubic time $O(n^3)$ on average over the inputs.
- D Worst case quadratic time $O(n^2)$, but expected linearithmic $O(n \log n)$

4. Design of algorithms

The country of Quadfinland is rightfully called the *Land of n Lakes* among the many tourists visiting every summer, who come to enjoy both nature and the safety of the excellent infrastructure. Quadfinland consists of $r \times c$ lakes connected in a grid-like fashion; so the total number of lakes is $n = rc$. Not only is Quadfinland full of lakes, it is also hilly, and lake v has an elevation[2] of $h(v)$ meters. (Let's agree that all $h(v)$ are integers.)

Going by boat from lake v to a neighbouring lake w entails using a complicated system of locks[3]; it takes $1 + |h(v) - h(w)|$ hours to travel directly from v to w. (Recall that $|x|$ the absolute value of x: $|-7| = |7| = 7$.)



In the above example, with $r = 2$ and $c = 3$, it takes $|3 - 1| + 1 = 2 + 1$ hours to traverse the canal from v_3 to v_6.

You are the newly-elected Minister of Lake Safety of Quadfinland. Where do you want your office located so that it has minum distance to all the other lakes? To be precise: at which lake u should your office be so that the maximum of the shortest distances to all other lakes is minimised? To be even more precise, and to introduce some notation, let V denote the set of all lakes (*vesistö* is the Finnish word for *waters*), and let $d(u, w)$ denote the minimum distance between two lakes. Determine $u \in V$ such that $\max_{v \in V} d(u, v)$ is minimised. If there's more than one optimal answer, any of them will do.

For example, in the above example, the fastest way from v_4 to v_6 is to travel via v_1, v_2, v_3, for a total time of

$$d(v_4, v_6) = 4 + |1 - (-1)| + |-1 - 2| + |2 - 1| + |3 - 1| = 12 \text{ hours.}$$

[2]**elevation**, *n.*: Height above a given level, especially sea level. Danish:*højde*.

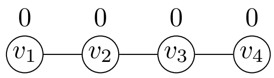
[3]**lock**, *n.*: A segment of a canal or other waterway enclosed by gates, used for raising and lowering boats between levels. Danish: *sluse*.

The best place for your office is lake v_6; it is at distance at most 28 to any other lake.

You can assume that input is give in the following form: First, a single line containing r and c. Then follow r lines each containing c integers: the elevations of the lakes the corresponding row. Your output should identify a single lake; let's agree that the lake names are numbered $v_1, \ldots, v_n$, left-to-right, bottom-to-top, as in our examples.
For example, in the above example, the input would be

```
2 3
1 30 3
-1 2 1
```

(a) *(1 pt.)* Assume $r = 1$ and Quadfinland is completely flat, so $h(v) = 0$ for every lake v, like this for $c = 4$:



Give an efficient algorithm.

State the running time of your algorithm in terms of the given parameters; ignore the time for reading the input. Faster is better. sol: +44

(b) *(2 pt.)* Assume $r = 1$. Give an efficient algorithm.

State the running time of your algorithm in terms of the given parameters; ignore the time for reading the input. Faster is better. sol: +44

(c) *(2 pt.)* Give an efficient algorithm.

State the running time of your algorithm in terms of the given parameters; ignore the time for reading the input. Faster is better. sol: +44

In all questions, use existing algorithms and data structures as much as you can, write clearly, and be brief. If you use the same algorithm for some of the questions, just write that, like "see 4c", instead of repeating yourself.