

# Algorithms and Data Structures

Exam 25 May 2022

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 1-4. 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  $f(N) \sim g(N)$ ?

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

☐  $f(N) = \sqrt{N^3} + \log N$  and  $g(N) = \sqrt{N} + \log N$

☒  $f(N) = N \log N$  and  $g(N) = (N \log N) - N$

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

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

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

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

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

☐  $f(N) = (\log N) \cdot (\log N) \cdot (\log N)$  and  $g(N) = (\log N)^2$

(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  $\sim 2N$

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

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

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

☐ A  $O(\log N)$

☐ B  $O(N)$

☒ C  $O(N \log N)$

☐ D  $O(N^2)$

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

```
# python3
i = 1
while i < N:
    j = N
    if i < 5:
        j = N * N
    while j > 0:
        stdio.write("*")
        j = j - 2
    i = 2 * i
```

```
// java
for (int i = 1; i < N; i = 2 * i) {
    int j = N;
    if( i < 5)
        j = N * N;
    while ( j > 0) {
        StdOut.print("*");
        j = j - 2;
    }
}
```

☐ A  $O(\log N)$

☐ B  $O(N)$

☒ C  $O(N \log N)$

☐ D  $O(N^2)$

- (f) (1 pt.) Find a recurrence relation for the number of arithmetic operations (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.)

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

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

☐ A  $T(N) = T(N-1) + 2$

☒ C  $T(N) = T(N-1) + T(N-2) + 5$

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

☐ D  $T(N) = 2 \cdot T(N-1) + 5$

- (g) (1 pt.) Assume I have a function  $f(\text{int } K)$  that runs in amortised constant time in  $K$ , but logarithmic worst case time in  $K$ . What is the running time of

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

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

(Choose the smallest correct estimate.)

☐ A Linearithmic in  $N$ .

☒ C Linear in  $N$ .

☐ B Amortised linear in  $N$ .

☐ D Impossible to say from the information given.

- (h) (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 = 1.0
for i in range(N):
    x = f(x) + f(x-1)
print(x)
```

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

float x = 1.0;
for (int i = 0; i < N; i = i + 1)
    x = f(x) + f(x-1);
System.out.println(x);
```

☒ A linear in  $N$

☐ B linearithmic in  $N$

☐ C quadratic in  $N$

☐ D even slower



```
1 import edu.princeton.cs.algs4.Merge;
2
3 class G {
4     int n;
5     int m;
6     Edge[] edges;
7
8     class Edge implements Comparable<Edge> {
9         int start;
10        int end;
11        public Edge(int start, int end) {
12            this.start = start;
13            this.end = end;
14        }
15
16        public int compareTo(Edge that) {
17            return this.start - that.start;
18        }
19    }
20
21    G(int num_nodes)
22    {
23        this.n = num_nodes;
24        this.edges = new Edge[0];
25        this.m = 0;
26    }
27
28
29    boolean has_edge(int start, int end) {
30        for (Edge e: this.edges)
31            if (e.start == start && e.end == end)
32                return true;
33        return false;
34    }
35
36
37    void add_edge(int start, int end) {
38        Edge[] new_edges = new Edge[this.m + 1];
39        for (int i = 0; i < this.m; ++i)
40            new_edges[i] = edges[i];
41        new_edges[m] = new Edge(start, end);
42        Merge.sort(new_edges);
43        this.edges = new_edges;
44        this.m += 1;
45    }
46 }
```

Figure 2: Class G, java version.

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

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

<pre># python3 g = G(10) g.add_edge(1, 4) g.add_edge(0, 1) print(g.has_edge(1,4)) print(g.has_edge(0,4)) print(g.has_edge(4,1))</pre>	<pre>// java G g = new G(10); g.add_edge(1, 4); g.add_edge(0, 1); println(g.has_edge(1,4)); println(g.has_edge(0,4)); println(g.has_edge(4,1));</pre>
---------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------

(b) (1 pt.) Clearly, class G is some kind of graph. Which one?

- |                                                      |                                                  |
|------------------------------------------------------|--------------------------------------------------|
| <input checked="" type="checkbox"/> A Directed graph | <input type="checkbox"/> B Undirected graph      |
| <input type="checkbox"/> C Weighted bipartite graph  | <input type="checkbox"/> D Minimum spanning tree |

(c) (1 pt.) Draw the data structure (not the graph!) after the following operations:

<pre># python3 g = G(10) g.add_edge(0, 1) g.add_edge(1, 4) g.add_edge(4, 0) g.add_edge(1, 2)</pre>	<pre>//java G g = new G(10); g.add_edge(0, 1); g.add_edge(1, 4); g.add_edge(4, 0); g.add_edge(1, 2);</pre>
----------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------

(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.)

(d) (1 pt.) Assume  $m > 1$ . What is the worst-case running time of `add_edge`?

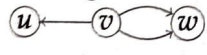
- |                                          |                                     |                                                       |                                     |
|------------------------------------------|-------------------------------------|-------------------------------------------------------|-------------------------------------|
| <input type="checkbox"/> A $O(\log m)$ . | <input type="checkbox"/> B $O(m)$ . | <input checked="" type="checkbox"/> C $O(m \log m)$ . | <input type="checkbox"/> D $O(1)$ . |
|------------------------------------------|-------------------------------------|-------------------------------------------------------|-------------------------------------|

(e) (1 pt.) Assume  $m > 1$ . What is the worst-case running time of `has_edge`?

- |                                          |                                                |                                            |                                     |
|------------------------------------------|------------------------------------------------|--------------------------------------------|-------------------------------------|
| <input type="checkbox"/> A $O(\log m)$ . | <input checked="" type="checkbox"/> B $O(m)$ . | <input type="checkbox"/> C $O(m \log m)$ . | <input type="checkbox"/> D $O(1)$ . |
|------------------------------------------|------------------------------------------------|--------------------------------------------|-------------------------------------|

(f) (1 pt.) What is the running time for creating a new graph with  $n > 1$  nodes and  $m = 2$  edges?

- |                                      |                                     |                                         |                                                |
|--------------------------------------|-------------------------------------|-----------------------------------------|------------------------------------------------|
| <input type="checkbox"/> A $O(nm)$ . | <input type="checkbox"/> B $O(n)$ . | <input type="checkbox"/> C $O(n + m)$ . | <input checked="" type="checkbox"/> D $O(1)$ . |
|--------------------------------------|-------------------------------------|-----------------------------------------|------------------------------------------------|

(g) (2 pt.) Write a method `outdegree( $v$ )` that returns the out-degree  $d_v$  of node  $v$ . (Recall that the outdegree is the number of edges from  $v$ , including duplicate edges. In particular, in the graph  we have  $d_v = 3$ .) Give the running time in terms of  $n$ ,  $m$ , and/or  $d_v$ . Faster is better. Pseudocode is fine; using existing library functions or data structures is fine; but you cannot change other parts of the data structure.

(h) (1 pt.) Let  $n \geq 2$  and  $k \geq 1$  be nonzero integers and consider the following operations:

```
# python3
g = G(n)
for _ in range(k):
    g.add_edge(0, 1)
```

```
//java
G g = new G(n);
for (int i = 0; i < k; i = i + 1)
    g.add_edge(0, 1);
```

How much space does the data structure use after these operations?

- ☐ A  $O(nk)$       ☐ B  $O(k)$       ☐ C  $O(k + n)$       ☐ D  $O(n)$

(i) (1 pt.) Describe how to change `add_edge` so that the original *datatype* `G` does not change (i.e., any sequence of `add_edge` and `has_edge` operations will give the same answers as before) but the operations from Question 2h use only  $O(1)$  space. You can ignore the effects on the extra method `outdegree` from `2g`.

(j) (1 pt.) Bob would like to speed up the query time. Which of his suggestions would significantly improve the running time of `has_edge`? (He is not interested in improving update time, but it shouldn't become much worse.)

- ☐ A Change the comparator of `Edge` so that it lexicographically compares first the start-values and then the end-values. Then use binary search instead of linear search in `has_edge`.
- ☐ B Use insertion sort instead of merge sort. This is because whenever a new edge is inserted, the other edges are already sorted, so insertion sort can be shown to run in linear time.
- ☐ C Use the array-resizing trick for the sequence of `Edges` as explained in [SW 1.3] for stacks. (In particular, only call mergesort when `g.m` is 2, 4, 8, 16, etc.) The performance improvement envisioned by Bob then holds in the *amortised* sense.
- ☐ D Store for each edge from  $u$  to  $v$  also the *opposite* edge (from  $v$  to  $u$ ) in another sequence, say `backwardsEdges`. Then for each query iterate over *both* sequences (edges and `backwardsEdges`), alternating between the two lists, until the query edge is found in both. (As before, if the edge is in neither list, return false.)

### 3. Operation of common algorithms and data structures.

(a) (1 pt.) Consider the following sequence of operations on a FIFO-queue, where a number  $i$  means `enqueue( $i$ )` and “\*” means `dequeue()`. The data structure is initially empty.

5 3 9 \* 4

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

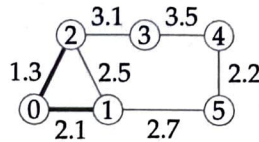
- ☒ A 3      ☐ B 4      ☐ C 5      ☐ D 9

(b) (1 pt.) Which comparison-based sorting algorithm sorts inputs of length  $N > 1$  in worst-case time  $O(N \log N)$ ?

- ☐ A Selection sort      ☐ B Insertion sort      ☒ C Mergesort      ☐ D Quicksort



(c) (1 pt.) Here is a graph in the middle of an execution of Prim'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 1-5

☐ D 4-5

(d) (1 pt.) Let me sort S E Q U E N C E using insertion sort. Which one of the following situations can arise at some time during the algorithm?<sup>1</sup>

☐ A C E Q U E N S E

☐ B N E Q U E S C E

☒ C E E Q S U N C E

☐ D E S C U E N Q E

(e) (1 pt.) Insert the 4 letters RATS (in that order) into a left leaning red-black binary search tree [SW 3.3] using alphabetic ordering. Draw the result.<sup>1</sup>

(f) (1 pt.) Consider the key-value pairs

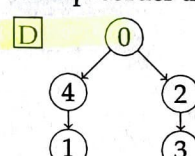
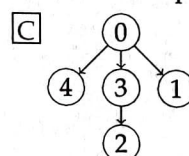
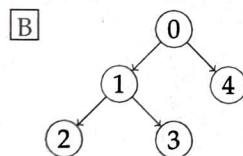
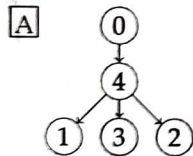
key	M	I	G	N	A	G
value	0	1	2	3	4	5

We use the hash function  $(\text{key}.\text{hashCode}() \& 0x7fffffff) \% 7$ . 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 linear probing. Draw the result.

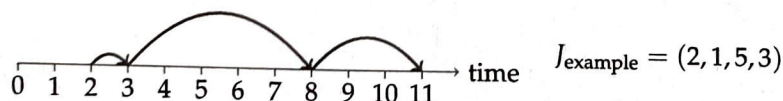
(g) (1 pt.) For which of the following trees can 0, 4, 1, 3, 2 not be a depth-first-search preorder traversal?



#### 4. Design of algorithms

Freddy Fireworks is a juggling robot specialising in juggling fireworks. His best-loved act is the *Globe Salute* – a ball-shaped firework which explodes in a breathtaking display of glitter and sparkles during his act.

Juggling basically consists of throwing and catching such a globe. A *juggling sequence* is a sequence  $J$  of  $m + 1$  many integers that indicate the number of seconds that the globe is in the air. For instance, the juggling sequence  $J_{\text{example}} = (2, 1, 5, 3)$  means: after 2 seconds, throw the globe so that it is airborne for 1 second, then catch it and throw it again (much higher), so that it stays airborne for 5 seconds, and so on. It takes 0 seconds for Freddy to throw-and-catch a globe. The first value of  $J$  indicates the time at which the sequence starts. For  $J_{\text{example}}$ , Freddy's actions can be drawn like this:



The fuse length  $F$  of a globe determines when it explodes. All globes are ignited at time 0. For instance, if  $F = 9$  then the example globe would explode during the third throw, while airborne.

<sup>1</sup> 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.