Exam - August 2022

Algorithms and Data Structures (DAT2, SW2, DV2)

Instructions. This exam consists of **five questions**, each divided into sub-questions. You must hand-in your solutions in digital exam as a **single pdf file**. You are encouraged to mark the multiple choice answers as well as the labelling of graphs directly in this exam sheet.

- Before starting solving the questions, read carefully the exam guidelines at `https://www.moodle.aau.dk/mod/page/view.php?id=1340499`.
- Read carefully the text of each exercise. Pay particular attention to the terms in bold.
- **CLRS** refers to the textbook T.H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition).
- You are allowed to refer to results in the textbook as well as exercise or self-study solutions posted in Moodle to support your arguments used in your answers.
- Make an effort to present your solutions neatly and precisely.

Question 1. 15 Pts

Identifying asymptotic notation. (Note: lg means logarithm in base 2)

(1.1) [5 Pts] Mark **ALL** the correct answers. $n\sqrt[2]{n} + \lg(n^2+n^2) + n^3 n \lg n$ is

☐ **a)** $O(n^5 \lg n)$ ☐ **b)** $\Omega(n^4)$ ☐ **c)** $\Theta(n^{4.5})$ ☐ **d)** $\Theta(n^3 \lg n)$ ☐ **e)** $O(n^5)$

(1.2) [5 Pts] Mark **ALL** the correct answers. $\lg 4^{\lg n} + \lg\lg n^{2000} - 100$ is

☐ **a)** $O(n)$ ☐ **b)** $\Omega(n)$ ☐ **c)** $O(n \lg n)$ ☐ **d)** $\Omega(\sqrt{n})$ ☐ **e)** $\Theta(\lg n)$

(1.3) [5 Pts] Mark **ALL** all the functions below that satisfy $\lg(f(n) \cdot g(m)) = \Theta(n+m)$.
Hint: CLRS p.52 Exercise 3.1–8

☐ **a)** $f(n) = n^2$, $g(m) = m^2$ ☐ **b)** $f(n) = 2^n$, $g(m) = 2^m$
☐ **c)** $f(n) = 2^n$, $g(m) = 4^{2m}$ ☐ **d)** $f(n) = n^2$, $g(m) = 2^m$

Solution 1.

(1.1)

$$n\sqrt[2]{n} + \lg(n^2+n^2) + n^3 n \lg n = n^{2.5} + n^2 + n^4 \lg n = \Theta(n^4 \lg n)$$

Therefore **a**, **b**, and **e** are correct.

(1.2)

$$\lg 4^{\lg n} + \lg\lg n^{2000} - 100 = \lg 2^{2\lg n} + \lg(2000 \lg n) - 100 = 2\lg n + \lg 2000 + \lg\lg n - 100 = \Theta(\lg n)$$

Therefore **a**, **c**, and **e** are correct.

(1.3) The correct answers are **b** and **c** as demonstrated below

$$\lg(n^2 \cdot m^2) = \lg n^2 + \lg m^2 = 2(\lg n + \lg m) = \Theta(\lg n + \lg m) \quad (\mathbf{a})$$
$$\lg(2^n \cdot 2^m) = \lg 2^n + \lg 2^m = n + m = \Theta(n+m) \quad (\mathbf{b})$$
$$\lg(2^n \cdot 4^{2m}) = \lg 2^n + \lg 2^{4m} = n + 4m = \Theta(n+m) \quad (\mathbf{c})$$
$$\lg(n^2 \cdot 2^m) = \lg n^2 + \lg 2^m = \lg n + m = \Theta(\lg n + m) \quad (\mathbf{d})$$

Question 2. 20 Pts

Answer the questions below concerning these two recurrences:

$$Q(n) = 4Q(n/2) + n^2\sqrt{n} \qquad \text{and} \qquad T(n) = T(n/4) + T(n/4) + \sqrt{n}\,.$$

Assume that $Q(n)$ and $T(n)$ are constant for sufficiently small n.
Remark: For each question, pay close attention to whether it concerns $Q(n)$ or $T(n)$.

(2.1) [5 Pts] Mark **ALL** correct answers.

- ☐ **a)** $Q(n)$ can be solved using Case 1 of the Master Theorem
- ☐ **b)** $Q(n)$ can be solved using Case 2 of the Master Theorem
- ☐ **c)** $Q(n)$ can be solved using Case 3 of the Master Theorem
- ☐ **d)** $Q(n)$ cannot be solved using the Master Theorem

(2.2) [5 Pts] Mark **ALL** correct answers.

- ☐ **a)** $Q(n) = \Omega(\lg n)$
- ☐ **b)** $Q(n) = O(n^3)$
- ☐ **c)** $Q(n) = \Omega(n)$
- ☐ **d)** $Q(n) = \Theta(n \lg n)$

(2.3) [10 Pts] Can we prove $T(n) = \Theta(\sqrt{n}\lg n)$ using the **master method**? If yes, then provide such a proof using the master method. If no, then argue why not.

Solution 2.

(2.1) Note that the recurrence is of the form $Q(n) = aQ(n/b) + f(n)$ where $a = 4$, $b = 2$, and $f(n) = n^{2+1/2}$. This recurrence falls into the third case of the Master Theorem (CLRS Thm. 4.1) because $f(n) = n^{2+1/2} = \Omega(n^{\log_b a+\epsilon}) = \Omega(n^{\log_2 4+\epsilon}) = \Omega(n^{2+\epsilon})$ for $0 < \epsilon < 1/2$. Moreover the "regularisation" condition $af(n/b) \le cf(n)$ holds for $c = 1/\sqrt{2}$ and $n \ge 0$. We prove the inequality below

$$\begin{aligned}
af(n/b) &= \\
&= 4 \cdot f(n/2) && (a = 4,\ b = 2) \\
&= 4 \cdot \left(\frac{n}{2}\right)^{2+1/2} && (f(n) = n^{2+1/2}) \\
&= 4 \cdot \frac{n^2\sqrt{n}}{4\sqrt{2}} \\
&= c \cdot f(n) && (c = 1/8,\ f(n) = n^2\sqrt{n})
\end{aligned}$$

By the Master Theorem (Case 3) we can conclude that $Q(n) = \Theta(f(n)) = \Theta(n^2\sqrt{n})$.

Therefore the only correct answer is **c**.

(2.2) As proven above, $Q(n) = \Theta(n^2\sqrt{n})$, therefore the correct answers are **a**, **b**, and **c**

(2.3) Note that the recurrence is of the form $T(n) = aT(n/b) + f(n)$ where $a = 2$, $b = 4$, and $f(n) = \sqrt{n}$. This recurrence falls into the second case of the Master Theorem (CLRS Thm. 4.1) because $f(n) = \sqrt{n} = n^{1/2} = \Theta(n^{\log_b a}) = \Theta(n^{1/2})$. Therefore, by the Master Theorem (Case 2) we can conclude that $T(n) = \Theta(n^{1/2}\lg n) = \Theta(\sqrt{n}\lg n)$.

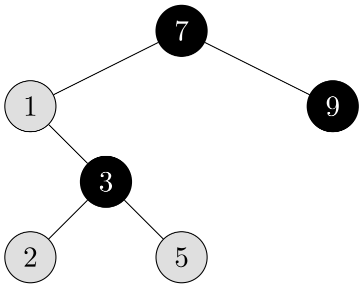
Question 3. 25 Pts

Understanding of known algorithms.

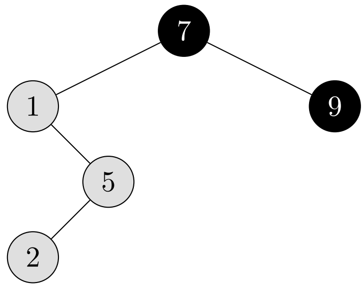
(3.1) [5 Pts] Mark **ALL** the correct statements.

- ☐ **a)** The adjacency matrix representation of a graph $G = (V, E)$ requires $\Theta(|V| + |E|)$ space.
- ☐ **b)** The BFS procedure works in place.
- ☐ **c)** $\textsc{Quick-Sort}$ sorts an array of n numbers in $O(n^2)$ time.
- ☐ **d)** $\textsc{List-Insert}$ and $\textsc{List-Delete}$ on linked-lists have the same asymptotic running-time.
- ☐ **e)** Running $\textsc{Tree-Successor}$ in a binary search tree with n elements takes $\Theta(\lg n)$ time.

(3.2) [4 Pts] Mark **ALL** the correct statements. Consider the binary trees T_1 and T_2 depicted below.



(T_1)



(T_2)

Remark: When answering **a**, **b**, and **c** do not consider the colours of the nodes in T_1 and T_2.

- ☐ **a)** Both T_1 and T_2 satisfy the binary search tree property
- ☐ **b)** T_1 is the result of $\textsc{Tree-Insert}(T_2, 3)$.
- ☐ **c)** T_2 is the result of $\textsc{Tree-Delete}(T_1, 3)$.
- ☐ **d)** Both T_1 and T_2 satisfy the red-black property.

(3.3) [6 Pts] Consider the hash table $H = 11, \textsc{Nil}, \textsc{Nil}, \textsc{Nil}, 70, 93, \textsc{Nil}, 29, 18, 63, \textsc{Nil}$. Insert the keys $32, 93, 10$ in H using *open addressing* with the auxiliary function $h'(k) = k$.

(i) Mark the hash table resulting by the insertion of these keys using **linear** probing.

☐ **a)** $11, \textsc{Nil}, 93, 10, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **b)** $11, 10, \textsc{Nil}, \textsc{Nil}, 70, 93, 93, 29, 18, 63, 32$

☐ **c)** $11, 10, 93, \textsc{Nil}, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **d)** none of the above

(ii) Mark the hash table resulting by the insertion of these keys using **quadratic** probing with $c_1 = 2$ and $c_2 = 7$.

☐ **a)** $11, 10, 93, \textsc{Nil}, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **b)** $11, \textsc{Nil}, 10, \textsc{Nil}, 70, 93, 93, 29, 18, 63, 32$

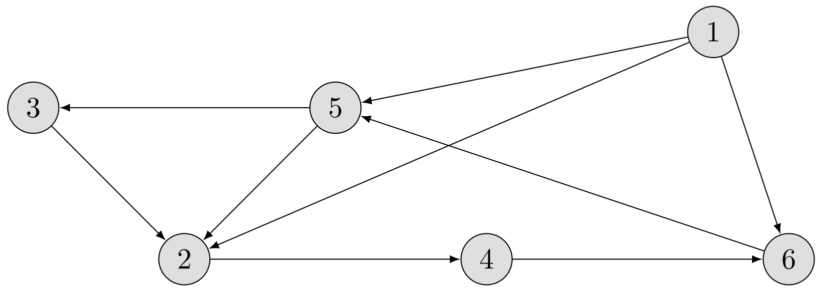
☐ **c)** $11, \textsc{Nil}, 10, 93, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **d)** none of the above

(iii) Mark the hash table resulting by the insertion of these keys using **double hashing** with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m-1))$.

☐ **a)** $11, \textsc{Nil}, 10, 93, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **b)** $11, 10, \textsc{Nil}, \textsc{Nil}, 70, 93, 93, 29, 18, 63, 32$

☐ **c)** $11, 10, 93, \textsc{Nil}, 70, 93, \textsc{Nil}, 29, 18, 63, 32$ ☐ **d)** none of the above

(3.4) [10 Pts] Consider the directed graph G depicted below.



(a) [2 Pts] Write the intervals for the discovery time and finishing time of each vertex in the graph obtained by performing a depth-first search visit of G (see CLRS Sec 22.3).
Remark: If more than one vertex can be chosen, choose the one with smallest vertex label.

(b) [2 Pts] Assign to each edge a label T (tree edge), B (back edge), F (forward edge), or C (cross edge) corresponding to the classification of edges induced by the DFS visit performed before.

(c) [4 Pts] Mark **ALL** the valid "parenthesis structures" of the discovery and finishing times in the sense of CLRS Theorem 22.7 resulting from some DFS visit performed on the above graph. *Remark:* In contrast with the above point, here vertices can be chosen arbitrarily.

☐ **a)** (1 (2 (4 (6 (5 (3 3) 5) 6) 4) 2) 1) ☐ **b)** (3 (2 (4 (6 (5 5) 6) 4) 2) 3) (1 1)

☐ **c)** (5 (2 (4 (6 6) 4) 2) (3 3) 5) (1 1) ☐ **d)** (1 1) (5 (2 (4 (6 6) 4) 2) (3 3) 5)

(d) [2 Pts] If G admits a topological sorting, then show the result of TOPOLOGICAL-SORT(G) (see CLRS Sec 22.4). If it doesn't admit a topological sorting, briefly argue why.

Solution 3.

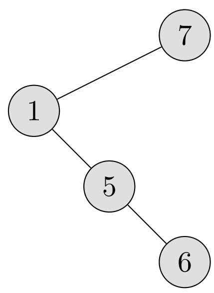
(3.1) **a)** Wrong. The adjacency matrix representation of a graph $G = (V, E)$ requires $\Theta(|V|^2)$ memory space.

b) Wrong. The BFS procedures maintains a queue which whose size is not constant in the size of the input graph.

c) Correct. QUICK-SORT worst-case running-time is $\Theta(n^2)$, hence it sorts an array of size n in $O(n^2)$.

d) Correct. LIST-INSERT and LIST-DELETE both run in $\Theta(1)$ time.

e) Wrong. TREE-SUCCESSOR runs in $\Theta(h)$ time where h is the heigh of the binary search tree. In case the tree is unbalanced $\Theta(h) \neq \Theta(\lg n)$. For example, finding the successor of the node 6 in the binary search tree below takes linear time in the size of the tree.



(3.2) **a)** Correct.

b) Wrong. The insertion of 3 in T_2 would place the node 3 as the right child of 2.

c) Correct. The deletion of 3 in T_3 would result in transplanting 5, which is the successor of 3, in place of 3.

d) Wrong. T_1 satisfies the red-black tree property, while T_2 doesn't (e.g., there are red nodes whose children are not black).

(3.3) Recall that the hash functions under linear probing, quadratic probing, and double hashing are

$$h(k,i) = (h'(k) + i) \mod m \qquad \text{(LINEAR PROBING)}$$
$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m \qquad \text{(QUADRATIC PROBING)}$$
$$h(k,i) = (h_1(k) + i h_2(k)) \mod m \qquad \text{(DOUBLE HASHING)}$$

i) The correct answer is **b**.

ii) The correct answer is **c**.

iii) The correct answer is **c**.

(3.4) The correct answers for (a) and (b) are depicted in the graph above. There, each vertex $v \in V$ is associated with the interval $[v.d, v.f]$ as computed by DFS, and each edge is labelled according to the corresponding classification.

(c) The correct answers are **a**, **b**, and **c**.

(d) The graph contains a cycle, namely $2 \to 4 \to 6 \to 5 \to 2$. Therefore it does not admit topological sorting. The presence of the cycle could also be spotted by the fact that the edge $(5, 2)$ was classified as a back-edge in (b).

Question 4. 20 Pts

Asymptotic runtime analysis.

Prof. Algo was asked to analyse the spread of Covid-19 within a company. To this end, he modelled the interactions of the all the workers of the company as a graph $G = (V, E)$ where each vertex $v \in V$ represents a worker and each edge $(u, v) \in E$ indicates frequent work interaction among u and v.

(a) [10 Pts] Assuming that the worker $s \in V$ has been infected, and that vaccinated people don't get infected, prof. Algo wants to determine how the virus can spreads in the company. To this end he devises the procedure SPREAD(G, s) that traverses the graph G marking all the workers of the company that can get (possibly indirectly) infected by the worker s.

SPREAD(G, s)

1 $s.infected$ = TRUE
2 let Q be an empty queue
3 ENQUEUE(Q, s)
4 **while** Q is not empty
5   u = DEQUEUE(Q)
6   **for each** $v \in G.Adj[u]$
7   **if** $\neg v.vaccinated \wedge \neg v.infected$
8             $v.infected$ = TRUE
9   ENQUEUE(Q, v)

Complete the following statements.
Remark: **ALL** the answers **must** be expressed as a function of either $|V|$, $|E|$ or both, where V and E are respectively the vertices and the edges of the input graph G.

a.1) The worst-case running time of lines 1–3 in Θ-notation is: $\Theta(1)$

a.2) In the worst-case, the **number of times** that line 5 is executed in O-notation is: $\Theta(V)$

a.3) In the worst-case, the **number of times** that line 7 is executed in O-notation is: $O(E)$

a.4) The worst-case running time of SPREAD(G, s) in O-notation is: $O(|V| + |E|)$

(b) [10 Pts] Let assume that G has vertex set $V = \{v_1, v_2, \dots, v_n\}$. Prof. Algo defined the *spread factor* of a worker $v_i \in V$ as the number of workers reachable from v_i in G (regardless of their vaccination status). The following procedure prints the workers of the company ordered in non-decreasing spread factor value.

PRINT-BY-SPREADFACTOR(G)

1 C = TRANSITIVE-CLOSURE(G)
2 let T be an empty red-black tree
3 **for** $i = 1$ **to** $|G.V|$
4     $v_i.key = -1$
5   **for** $j = 1$ **to** $|G.V|$
6         $v_i.key = v_i.key + C[i, j]$
7   RB-INSERT(T, v_i)
8 INORDER-TREE-WALK$(T.root)$

Complete the following statements.
Remark: **ALL** the running times **must** be expressed as a function of either $|V|$, $|E|$ or both, where V and E are respectively the vertices and the edges of the input graph G.

b.1) The worst-case running time of line 1 in Θ-notation is: $\Theta(|V|^3)$

b.2) The worst-case running time of line 2 in Θ-notation is: $\Theta(1)$

b.3) The worst-case running time of lines 3–7 in Θ-notation is: $\Theta(|V|^2)$

b.4) The worst-case running time of PRINT-BY-SPREADFACTOR(G) in Θ-notation is: $\Theta(|V|^3)$

Question 5. 20 Pts

Solving computational problems.

Consider the $n \times n$ matrix $W = (w_{ij})$ representing the edge weights of a n-vertex complete directed graph $G = (V, E)$, where vertices are conveniently numbered $1, 2, \ldots, n$. Assume that each entry w_{ij} represents the probability to move from the vertex i to the vertex j in one step. In this setting, a path $\pi = v_1, v_2, \ldots, v_T$ of length $T \geq 1$ can be interpreted as a *random walk*.

A random walk $\pi = v_1, v_2, \ldots, v_T$ is said to be **monotone** if the corresponding vertices are non-decreasing, i.e., $v_1 \leq v_2 \leq \cdots \leq v_T$. Given as input a vertex v and a length $T \geq 1$, prof. Algo developed a procedure called MonoWalk(W, v, T) that calculates the probability that by moving T steps forward in the graph G from the vertex v, the random walk that is traversed is monotone. Prof. Algo's implementation of MonoWalk(W, v, T) returns the value $P(v, 1)$ calculated according to the recurrence

$$P(i,t) = \begin{cases} 1 & \text{if } t = T \\ \sum_{j=i}^{n} P(j, t+1) \cdot w_{ij} & \text{if } 1 \leq t < T. \end{cases}$$

Unfortunately, a naive implementation of the above recurrence led to a very slow algorithm. Help Prof. Algo enhance his algorithm by using the **dynamic programming** algorithm principle.

(a) [10 Pts] Describe a **top-down** dynamic programming implementation for MonoWalk(W, v, T)

(b) [10 Pts] Describe a **bottom-up** dynamic programming implementation for MonoWalk(W, v, T)

Remarks:

- To answer the questions you don't need neither to understand probabilities, nor to understand why Prof. Algo's implementation is correct.
- To help yourself understanding how the recurrence P unravels, you can calculate $P(1, 1)$ for $T = 3$ and the graph G with weight matrix

$$W = \begin{pmatrix} 0.1 & 0.3 & 0.6 \\ 0.4 & 0.2 & 0.2 \\ 0.7 & 0 & 0.3 \end{pmatrix}.$$

- The description of the algorithmic procedures must be given **both** by providing the pseudocode and by explaining in detail how it works.
- If you want, you can use auxiliary procedures to solve the above task. Remember you have to provide their pseudocode too.

Solution 5.

(a) A top-down dynamic programming implementation for MonotoneWalk(W, v, T) is

```
MonotoneWalk(W, v, T)
let n be the number of vertices of W
let P[1..n, 1..T] be a new array
// initialise all entries as not visited
for t = 1 to T
      for i = 1 to n
            P[i, t] = -1
// call the memoized implementation of the recurrence P
return MemoizedP(W, P, v, 1)
```

```
MemoizedP(W, P, i, t)
if P[i, t] < 0
    // If not visited yet
    if t = T
        P[i, t] = 1
    else
        P[i, t] = 0
        for j = i to n
            P[i, t] = P[i, t] + MemoizedP(W, P, j, t + 1) · W[i, j]
return P[i, t]
```

(b) A bottom-up dynamic programming implementation for MonotoneWalk(W, v, T) is

```
MonotoneWalk(W, v, T)
let n be the number of vertices of W
let P[1 . . n, 1 . . T] be a new array
for i = 1 to n
    P[i, T] = 1
for t = T − 1 downto 1
    for i = 1 to n
        P[i, t] = 0
        for j = i to n
            P[i, t] = P[i, t] + P[j, t + 1] · W[i, j]
return P[v, 1]
```