Exam - June 2021

Algorithms and Data Structures

Instructions. This exam consists of **five questions** and you have time until 13:00 to submit your solution in digital exam. You can answer the questions directly on this paper, or use additional sheets of paper which have to be hand-in 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=1173709`.
- Read carefully the text of each exercise. Pay particular attentions 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 some arguments used in your answers.
- Make an effort to use a readable handwriting and to present your solutions neatly.

Question 1. 15 Pts

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

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

☐ **a)** $\Theta(n^5 \lg n)$ ☐ **b)** $\Theta(n)$ ☐ **c)** $\Theta(n^{2.5})$ ☐ **d)** $\Theta(n^5 \lg n^5)$ ☐ **e)** $\Theta(n^5)$

(1.2) [5 Pts] Mark **ALL** the correct answers. $n^2\sqrt{n} + n \log_3 2^n$ is

☐ **a)** $\Theta(n^5)$ ☐ **b)** $\Omega(n)$ ☐ **c)** $\Theta(n^{2.5})$ ☐ **d)** $\Omega(\sqrt{n})$ ☐ **e)** $O(n^5)$

(1.3) [5 Pts] Mark **ALL** the correct answers. $100 \cdot n^2 + n^2 \lg 8^n + \frac{n \lg n}{0.5} + \lg n^n$ is:

☐ **a)** $\Omega(n \lg n)$ ☐ **b)** $O(n^3)$ ☐ **c)** $O(n^2)$ ☐ **d)** $\Omega(n^2 \lg n)$ ☐ **e)** $O(n^2 \lg n)$

Solution 1.

(1.1)

$$n^2\sqrt{n} + n^5 \lg n^5 + n \lg 2^n = n^{2.5} + 5n^5 \lg n + n^2 = \Theta(n^5 \lg n)$$

Therefore **a** and **d** are correct.

(1.2)

$$n^2\sqrt{n} + n \log_3 2^n = n^{2.5} + n^2 \log_3 2 = \Theta(n^{2.5})$$

Therefore **b**, **c**, **d**, and **e** are correct.

(1.3)

$$100 \cdot n^2 + n^2 \lg 8^n + \frac{n \lg n}{0.5} + \lg n^n = 100 \cdot n^2 + 3n^3 + 2n \lg n + n \lg n = \Theta(n^3)$$

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

Question 2. 20 Pts

Consider the following recurrences

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 0 \\ n \cdot T(n-1) & \text{if } n > 0 \end{cases} \qquad Q(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8 \cdot Q(n/2) + 2^n & \text{if } n > 1 \end{cases}$$

Answer the questions below concerning these two recurrences. For each question, play 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)** $T(n)$ can be solved using the Master Theorem

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

- ☐ **a)** $Q(n) = \Theta(2^n \lg n)$
- ☐ **b)** $Q(n) = O(n^3)$
- ☐ **c)** $Q(n) = \Theta(2^n)$
- ☐ **d)** $Q(n) = \Omega(n^{100})$

(2.3) [10 Pts] Prove that $T(n) = \Omega(2^n)$ using the substitution method.

Solution 2.

(2.1) In the next point we show that $Q(n)$ can be solved using the Case 3 of the Master Theorem. In contrast, the recurrence $T(n)$ does not comply with the format required for the Master Theorem. Therefore **c** is the only correct answer.

(2.2) Note that the recurrence is of the form $Q(n) = aQ(n/b)+f(n)$ where $a = 8$, $b = 2$, and $f(n) = 2^n$. We can solve the recurrence using the master method. This recurrence falls into the third case, because $f(n) = 2^n = \Omega(n^{3+\epsilon}) = \Omega(n^{\log_b a+\epsilon})$ for any $\epsilon > 0$. Moreover the "regularisation" condition $af(n/b) \leq cf(n)$ holds for $c = 1$ and $n \geq 6$. We prove the inequality below

$$\begin{aligned} af(n/b) &\leq cf(n) \\ 8 \cdot 2^{n/2} &\leq 2^n && (a = 8,\ b = 2,\ c = 1, \text{ and } f(n) = 2^n) \\ 2^{3+n/2} &\leq 2^n && (8 = 2^3) \\ 3 + n/2 &\leq n && (\text{apply lg on both sides}) \\ 3 &\leq n/2 && (\text{subtract } n/2 \text{ on both sides}) \\ 6 &\leq n\,. && (\text{multiply by 2 on both sides}) \end{aligned}$$

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

Therefore, the correct answers are **c** and **d**.

(2.3) We rewrite $T(n)$ making explicit the constants hidden behind the Θ notation:

$$T(n) = \begin{cases} d & \text{if } n = 0 \\ n \cdot T(n-1) & \text{if } n > 0 \end{cases}$$

for some $d > 0$. Note that we have already seen a similar recurrence in Exercise Session 4, namely Exercise 4.a. One can readily see that $T(n) = d \cdot n!$. Therefore, to prove that $T(n) = \Omega(2^n)$, it suffices to show that $n! = \Omega(2^n)$. In what follows we recall the proof given in Exercise Session 4, however for the exam it suffices to mention that we already know that $n! = \Omega(2^n)$.

To prove that $n! = \Omega(2^n)$ we show that for all $n \geq 1$, $n! \geq c2^n$ for some suitable constant $c > 0$ (notice that this corresponds to chose $n_0 = 1$ in the definition of Ω-notation).

Base Case $(n = 1)$**.** $n! = 1! = 1 \geq 2 = 2^n$. Thus, for $n = 1$, $n! \geq c2^n$ holds when $c \geq 1/2$.
Inductive Step $(n > 1)$**.** We have that

$$
\begin{aligned}
n! &= n \cdot (n-1)! && \text{(def. factorial)} \\
&\geq n \cdot c2^{n-1} && \text{(inductive hypothesis)} \\
&\geq 2 \cdot c2^{n-1} && (n \geq 2) \\
&= c2^n\,. && \text{(choosing } c \geq 1/2\text{)}
\end{aligned}
$$

Thus, for $c \geq 1/2$ we have that $n! \geq c2^n$ for all $n \geq 1$ from which we conclude $n! = \Omega(2^n)$.

Question 3. 25 Pts

Understanding of known algorithms.

(3.1) [5 Pts] Mark **ALL** the correct statements. Consider a modification to QUICKSORT, called MAXQUICKSORT, such that each time PARTITION is called, the maximum element of the sub-array to partition is found and used as a pivot.

- ☐ **a)** MAXQUICKSORT best-case running time is $\Theta(n^2)$
- ☐ **b)** If A is already sorted, then the running time of MAXQUICKSORT(A) is $\Theta(n \lg n)$
- ☐ **c)** MAXQUICKSORT(A) sorts the array A in **non-increasing** order
- ☐ **d)** MAXQUICKSORT worst-case running time is $O(n^3)$
- ☐ **e)** MAXQUICKSORT works in-place

(3.2) [4 Pts] Mark **ALL** the correct statements. Consider the array $A = [4, 3, 6, 2, 1, 5]$ and assume that $A.\mathit{heap\text{-}size} = A.\mathit{length}$.

- ☐ **a)** The binary tree interpretation of A satisfies the binary search tree property
- ☐ **b)** The result of MAX-HEAPIFY(A, 1) is $[6, 3, 5, 2, 1, 4]$
- ☐ **c)** The result of MAX-HEAPIFY(A, 1) is $[6, 3, 4, 2, 1, 5]$
- ☐ **d)** A satisfies the max-heap property

(3.3) [6 Pts] Consider the hash table $H = 97, \text{NIL}, \text{NIL}, 14, \text{NIL}, \text{NIL}, \text{NIL}, 29, \text{NIL}, 75, 32$. Insert the keys $55, 8, 10$ in H using *open addressing* with the auxiliary function $h'(k) = k$.

Mark the hash table resulting by the insertion of these keys using linear probing.

- ☐ **a)** $97, \text{NIL}, \text{NIL}, 14, \text{NIL}, 10, 55, 29, 8, 75, 32$
- ☐ **b)** $97, 55, 10, 14, \text{NIL}, \text{NIL}, \text{NIL}, 29, 8, 75, 32$
- ☐ **c)** $97, 10, \text{NIL}, 14, \text{NIL}, \text{NIL}, 55, 29, 8, 75, 32$
- ☐ **d)** none of the above

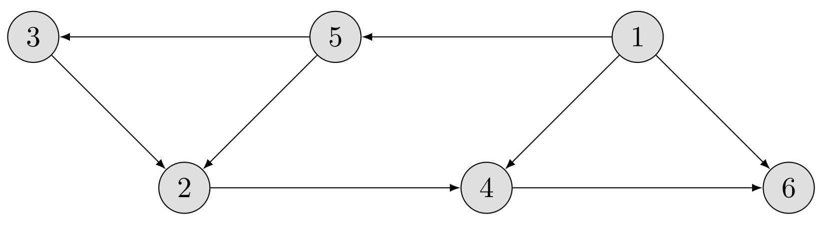
Mark the hash table resulting by the insertion of these keys using quadratic probing with $c_1 = 2$ and $c_2 = 4$.

- ☐ **a)** $97, \text{NIL}, \text{NIL}, 14, \text{NIL}, 10, 55, 29, 8, 75, 32$
- ☐ **b)** $97, 55, 10, 14, \text{NIL}, \text{NIL}, \text{NIL}, 29, 8, 75, 32$
- ☐ **c)** $97, 10, \text{NIL}, 14, \text{NIL}, \text{NIL}, 55, 29, 8, 75, 32$
- ☐ **d)** none of the above

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)** $97, \text{NIL}, \text{NIL}, 14, \text{NIL}, 10, 55, 29, 8, 75, 32$
- ☐ **b)** $97, 55, 10, 14, \text{NIL}, \text{NIL}, \text{NIL}, 29, 8, 75, 32$
- ☐ **c)** $97, 10, \text{NIL}, 14, \text{NIL}, \text{NIL}, 55, 29, 8, 75, 32$
- ☐ **d)** none of the above

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



(a) 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 label.

(b) Mark the corresponding "parenthesization" of the vertices in the sense of CLRS Theorem 22.7 resulting from the DFS visit performed before

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

☐ **c)** (1 (4 4) (5 (2 2) (3 3) 5) (6 6) 1) ☐ **d)** none of the above

(c) 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.

(d) 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) MAXQUICKSORT selects as a pivot element the max element in the subarray. The particular choice of the pivot makes the partition become unbalanced, leading to an algorithm that performs both the best-case and worst-case in $\Theta(n^2)$ time. Finding the max element can be easily implemented *in-place* therefore MAXQUICKSORT, like QUICKSORT, works in-place. The particular choice of the pivot element does not change the fact that MAXQUICKSORT, like QUICKSORT sorts the given array in non decreasing order.

Therefore the correct answers are **a**, **d**, and **e**.

(3.2) The only correct answer is **b**.

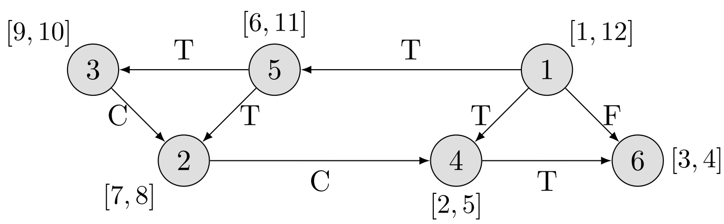
(3.3) The resulting hash tables are respectively:

linear probing: $97, 55, 10, 14,$ NIL, NIL, NIL, $29, 8, 75, 32$

quadratic probing: $97,$ NIL, NIL, $14,$ NIL, $10, 55, 29, 8, 75, 32$

double hashing: $97, 10,$ NIL, $14,$ NIL, NIL, $55, 29, 8, 75, 32$

(3.4) The correct answers for (a) and (c) are depicted in the graph below. 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.



(b) the corresponding "parenthesization" of the vertices is (1 (4 (6 6) 4) (5 (2 2) (3 3) 5) 1). Therefore the correct answer is **b**.

(d) The graph is acyclic –this can be seen by the absence of *back* edges in the DFS classification– therefore it admits topological sorting. TOPOLOGICAL-SORT(G) prints the vertices by decreasing order of finishing time, leading to $\langle 1, 5, 3, 2, 4, 6 \rangle$.

Question 4. 20 Pts

Asymptotic runtime analysis.

Prof. Algo has been asked to analyse user interactions in a social network. Prof. Algo started by modelling the social network as a graph $G = (V, E)$ where each vertex represents a user of the network and there exists an edge $(u, v) \in E$ if and only if user v liked some content posted by user u. Additionally, G is equipped with a weight function $w\colon E \to \mathbb{N}$ such that, for $(u, v) \in E$, $w(u, v)$ is the number of likes given by user v to user u.

(a) [10 Pts] Interested in discovering groups of users having intense mutual interactions, Prof. Algo defines the concept of *k-ranked group* as a strongly connected component $C \subseteq V$ in the subgraph $G^k = (V, E^k)$ where $E^k = \{(u, v) \in E \mid w(u, v) \geq k\}$. Then, he provides the following algorithm to find all *k-ranked groups* of G.

RANKEDGROUPS(G, w, k)

1 Let G^k be an empty graph.
2 $G^k.V = G.V$
3 **for each** $u \in G.V$
4   let $G^k.Adj[u]$ be an empty list
5   **for each** $v \in G.Adj[u]$
6   **if** $w(u, v) \geq k$
7   LIST-INSERT$(G^k.Adj[u], v)$
8 STRONGLY-CONNECTED-COMPONENTS(G^k)

Perform an asymptotic analysis of the worst-case running time of RANKEDGROUPS(G, w, k). Motivate your answer.

(b) [10 Pts] Prof. Algo defines the *influence* of an user $u \in V$ as *influence*$(u) = \sum_{v \in V} \delta(u, v)$, where $\delta(u, v)$ is the shortest path weight from u to v in G. Then, he provides the following algorithm which prints the vertices of the graph in non-decreasing order of influence.

PRINTBYINFLUENCE(G, w)

Let T be an empty binary search tree
for each $s \in V$
    DIJKSTRA(G, w, s)
    $s.key = 0$
    for each $v \in V - \{s\}$
        $s.key = s.key + v.d$
    TREE-INSERT(T, s)
INORDER-TREE-WALK$(T.root)$

Perform an asymptotic analysis of the worst-case running time of PRINTBYINFLUENCE(G, w). Motivate your answer.

Solution 4.

(a) The worst-case running time of RANKEDGROUPS(G, w, k) occurs when all edges in G have weight greater then or equal to k. The construction of the graph G^k (lines 1–7) overall takes $\Theta(V + E)$, and the call to STRONGLY-CONNECTED-COMPONENTS(G^k) takes $\Theta(V + E)$ (see CLRS sec. 22.5) because in the worst-case the size of G^k is equal to that of G. Thus, RANKEDGROUPS(G, w, k) worst-case running time is $\Theta(V + E)$.

(b) The worst-case running time of PRINTBYINFLUENCE(G, w) is $O(V^3)$. Indeed if we use Dijkstra's algorithm with the linear-array implementation of the min-priority queue, $|V|$ calls of DIJKSTRA algorithm take $O(V^3 + VE) = O(V^3)$. As before, the sequential insertion of $|V|$ elements in the binary search tree, may take in the worst-case $\Theta(V^2)$, and the final call to INORDER-TREE-WALK takes linear time in the number of elements in the tree, that is $\Theta(V)$.

Question 5. 20 Pts

Solving computational problems.

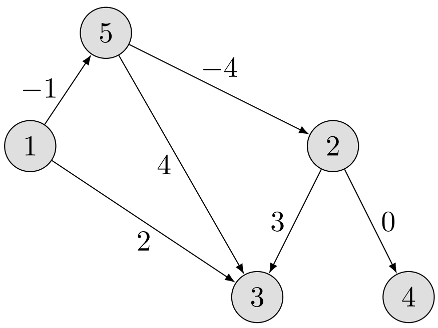
Given a directed **acyclic** graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and weight function $w\colon E \to \mathbb{R}$, we consider the problem of finding a **longest** (maximally-weghted) simple path from i to j for all pairs of vertices $i, j \in V$.

(a) [10 Pts] Describe a **bottom-up** dynamic programming procedure $\textsc{AllPairsLongestPath}(G, w)$ that returns an $n \times n$ matrix $L = (l_{ij})_{i,j \in V}$ where l_{ij} is the weight of a longest simple path from i to j.

(b) [10 Pts] Describe a procedure $\textsc{PrintLongestPath}(G, w, i, j)$ that prints a longest simple path from i to j.

Remarks:

- The description of the algorithmic procedures must be given **both** by providing the pseudocode and by explaining in detail how it works.
- Specify in your solution whether the weighted graph (G, w) is assumed to be represented using adjacency matrix or adjacency lists.
- Try to execute your algorithm on the following example. You may catch some errors you did not foresee while designing your algorithm.



L	1	2	3	4	5
1	0	-5	3	-5	-1
2	$-\infty$	0	3	0	$-\infty$
3	$-\infty$	$-\infty$	0	$-\infty$	$-\infty$
4	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$
5	$-\infty$	-4	4	-4	0

For instance, the longest path from vertex 1 to vertex 3 is the sequence $1, 5, 3$ having weight $-1+4 = 3$, while the longest path from 2 to 1 is the empty sequence with weight $-\infty$.

Solution 5.

(a) Since G is acyclic all paths are simple paths. We will present two solutions: the first one works similarly to the $\textsc{Floyd-Warshall}$ algorithm (see CLRS p.695); the second one, in the same line as $\textsc{Dag-Shortest-Paths}$ (CLRS p.655), exploits a topological sorting of G.

First Solution. We represent (G, w) using the adjacency matrix $W = (w_{ij})$ where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ -\infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

For $0 \leq k \leq n$, we define the weight of a longest simple path from v_i to v_j using intermediate vertices $\{v_1, \ldots, v_k\}$ as follows

$$l_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \max(l_{ij}^{(k-1)}, l_{ik}^{(k-1)} + l_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Because for any path all intermediate vertices are in the set $\{v_1, \ldots, v_n\}$, the matrix $L^{(n)} = (l_{ij}^{(n)})$ gives the final answer, that is $l_{ij}^{(n)}$ is weight of a longest simple path from v_i to v_j. We can compute $L^{(n)}$ bottom-up as follows.

AllPairsLongestPath(W)

1 $n = W.rows$
2 $L^{(0)} = W$
3 **for** $k = 1$ **to** n
4 Let $L^{(k)} = (l_{i,j}^{(k)})$ be a new $n \times n$ matrix
5 **for** $i = 1$ **to** n
6   **for** $j = 1$ **to** n
7    $l_{ij}^{(k)} = \max(l_{ij}^{(k-1)}, l_{ik}^{(k-1)} + l_{kj}^{(k-1)})$
8 **return** $L^{(n)}$

The worst-case running time of the above algorithm is $\Theta(n^3)$, due to the 3 nested for-loops in lines 3–7.

Second Solution. This solution works by repeatedly applying a variant of Dag-Shortest-Paths (CLRS p.655) to compute the longest simple paths. For this algorithm we represent (G, w) using the adjacency lists.

AllPairsLongestPath(G, w)

1 $n = |G.V|$
2 Let $L = (l_{ij})$ be a new $n \times n$ matrix initialised with $-\infty$
3 **for** $i = 1$ **to** n
4 $l_{ii} = 0$
5 topologically sort the vertices of G
6 **for each** vertex $i \in V$
7 **for each** vertex $k \in V$, taken in topologically sorted order
8   **for each** vertex $j \in G.Adj[k]$
9    $l_{ij} = \max(l_{ij}, l_{ik} + w(k, j))$
10 **return** L

The running-time of the above algorithm is $\Theta(|V||E| + |V|^2)$. Indeed, the initalisation of L costs $\Theta(|V|^2)$, topologically sorting the vertices of G takes $\Theta(|V| + |E|)$ (see CLRS section 22.5), and the nested for-loops in lines 6–9 is $\Theta(|V|(|V| + |E|)) = \Theta(|V||E| + |V|^2)$.

(b) To print the longest paths from i to j we first construct a predecessor graph, then we use the procedure Print-Path(G, i, j) (see CLRS p.601) to print the path. This will be done similarly to Dag-Shortest-Paths (CLRS p.655), assuming (G, w) is represented using adjacency lists.

Print-Longest-Path(G, w, i, j)

1 **for each** $v \in G.V$
2 $v.l = -\infty$
3 $v.\pi = $ Nil
4 $i.l = 0$
5 topologically sort the vertices of G
6 **for each** vertex $u \in V$, taken in topologically sorted order
7 **for each** vertex $v \in G.Adj[u]$
8   **if** $v.l < u.l + w(u, v)$
9    $v.l = u.l + w(u, v)$
10    $v.\pi = u$
11 Print-Path(G, i, j)

The running-time of Print-Longest-Path is $\Theta(|V| + |E|)$. Indeed the initialisation of the vertex attributes in lines-1–4 takes $\Theta(|V|)$, topologically sorting the vertices of G takes $\Theta(|V| + |E|)$ (see CLRS section 22.5), the nested for-loops in lines 6–10 take $\Theta(|V| + |E|)$, and the call to Print-Path in line 11 takes $\Theta(|V|)$ (see CLRS p.601).