Re-Exam - August 2020

Algorithms and Data Structures (DAT2, SW2, DV2)

Instructions. This exam consists of **five questions** and you have **four hours** to solve them. 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**.

- Before starting solving the questions, read carefully the exam guidelines at `https://www.moodle.aau.dk/mod/page/view.php?id=1048323`.
- Read carefully the text of each exercise before solving it! 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).
- Make an effort to use a readable handwriting and to present your solutions neatly.

Question 1. 19 Pts

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

(1.1) [3 Pts] Mark **ALL** the correct answers. $\lg n^{100} + \sqrt{n} + \lg 8^n$ is

☐ **a)** $\Theta(\lg n)$ ☐ **b)** $\Theta(n)$ ☐ **c)** $\Theta(\sqrt{n})$ ☐ **d)** $\Theta(n^2)$ ☐ **e)** $\Theta(2^n)$

(1.2) [3 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 32^n + 100n^2$ is

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

(1.3) [3 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 5^n + 100n^4$ is

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

(1.4) [10 Pts] Mark **ALL** the correct answers. Consider the following recurrence

$$T(n) = \begin{cases} 60 & \text{if } n = 1 \\ T(3n/4) + \sqrt{n} & \text{if } n > 1 \end{cases}$$

☐ **a)** Using the second case of the master method one can prove $T(n) = \Theta(n^2)$

☐ **b)** The master method cannot be used to solve the above recurrence

☐ **c)** Using the first case of the master method one can prove $T(n) = \Theta(\sqrt{n})$

☐ **d)** Using the master method one can prove that $T(n) = O(n^2)$.

☐ **e)** Using third case of the master method one can prove $T(n) = \Theta(\sqrt{n})$

Solution 1.

(1.1) $\lg n^{100} + \sqrt{n} + \lg 8^n = 100 \lg n + \sqrt{n} + 3n = \Theta(n)$.

Therefore the only correct answer is (b).

(1.2) $n \lg n^2 + \lg 32^n + 100n^2 = 2n \lg n + 5n + 100n^2 = \Theta(n^2)$.

Therefore the correct answers are (d), and (e).

(1.3) $n \lg n^2 + \lg 5^n + 100n^4 = 2n \lg n + n \lg 5 + 100n^4 = \Theta(n^4)$.

Therefore the correct answers are (a), (b), (c), and (d).

(1.4) This question is taken from an old exam which was given in preparation for the exam.

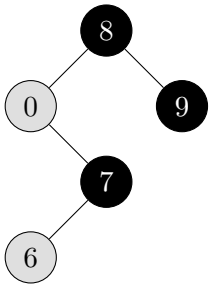
The recurrence is in the format $T(n) = aT(n/b) + f(n)$ where $a = 1$, $b = 4/3$, and $f(n) = \sqrt{n}$. We have that $f(n) = n^{0.5} = \Omega(n^{(\log_{4/3} 1)+\epsilon}) = \Omega(n^{\epsilon})$ for some constant $\epsilon > 0$ (e.g., $\epsilon = 0.5$). The regularity condition $af(n/b) \leq cf(n)$ is satisfied choosing some constant $b \leq c \leq 1$, since $af(n/b) = \sqrt{n} \cdot \sqrt{b}$. Therefore we can use the third case of the Master method which tell us the solution is $T(n) = \Theta(f(n)) = \Theta(\sqrt{n})$.

Having proved that $T(n) = \Theta(\sqrt{n})$ we can also conclude that $T(n) = O(n^2)$ thanks to the fact that n^2 grows asymptotically faster than $\sqrt{n}$.

Therefore the correct answers are (d) and (e).

Question 2. 21 Pts

(2.1) [3 Pts] Mark **ALL** the correct statements. Consider the tree T depicted below



- ☐ **a)** The height of T is 4
- ☐ **b)** T satisfies the binary-search tree property
- ☐ **c)** T satisfies the red-black tree property (NIL nodes are omitted)
- ☐ **d)** T satisfies the max-heap property
- ☐ **e)** T corresponds to the array $[8, 0, 9, 7, 6]$ interpreted as a binary tree.

(2.2) [5 Pts] Mark **ALL** the correct statements. Consider a modification to MERGE-SORT(A, p, r), called MI-SORT, that when $p < r$ and $r - p \leq 5$ calls INSERTION-SORT on the subarray $A[p\,..\,r]$, otherwise it works as MERGE-SORT.

- ☐ **a)** MI-SORT worst-case running time is $\Theta(n^2)$
- ☐ **b)** MI-SORT worst-case running time is $\Theta(n \lg n)$
- ☐ **c)** MI-SORT best-case running time is $\Theta(n \lg n)$
- ☐ **d)** MI-SORT best-case running time is $\Theta(n)$
- ☐ **e)** MI-SORT works in-place

(2.3) [13 Pts] Consider the following algorithm to find the greatest element in an array. It takes a non empty array $A[1\,..\,A.length]$, two indices p and q such that $1 \leq p \leq q \leq A.length$, and returns the value of the greatest element in the subarray $A[p\,..\,q]$.

```
GREATEST(A, p, q)
 1  if q = p
 2       return A[p]
 3  else
 4       m = ⌊(p + q)/2⌋
 5       maxl = GREATEST(A, p, m)
 6       maxr = GREATEST(A, m + 1, q)
 7       if maxl < maxr
 8            return maxr
 9       else
10            return maxl
```

In the following we consider $n = q - p$ to be the size of the problem. You can assume that $n = 2^k$ for some $k \in \mathbb{N}$.

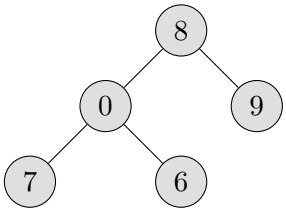
(a) [3 Pts] Write the recurrence $T(n)$ describing the running time of GREATEST. As usual, you may use constants $c_1, c_2, \ldots$ to indicate the running time of elementary statements.

(b) [10 Pts] Prof. Algo claims that GREATEST runs in $\Theta(n)$. Prove that Prof. Algo's claim is correct by solving the recurrence $T(n)$.

Solution 2.

(2.1) The height of the tree is 3 since its longest path from the root to a leaf has 3 edges. The tree satisfies the binary search tree property as well as the red-black tree property under the assumption that NIL nodes are omitted. T does **not** satisfy the max-heap property because for instance the node 9 is a child than the node 8.

Finally, the binary tree interpretation of the array $[8, 0, 9, 7, 6]$ is



Therefore, the correct answers are (b) and (c).

(2.2) We know that MERGE-SORT does not work in place, therefore neither MI does. The asymptotic running time of MI does not differ from that of MERGE-SORT because the cost of running INSERTION-SORT on arrays of size bounded by 5 can be treated as a constant both for the best-case and the worst-case running time analysis —indeed the running time can be bounded both from above and below by a constant.

Therefore the correct answers are (b) and (c).

(2.3) The solution of this question is almost identical to an exercise given in Exercise Session 3.

(a) The running time of GREATEST is described by the following recurrence

$$T(n) = \begin{cases} c_1 & \text{if } n \leq 1, \\ 2T(n/2) + c_2 & \text{if } n > 1. \end{cases}$$

where ${}_c1$ is a positive constant representing the running time for lines 1–2, and c_2 is another positive constant representing the aggregated running time for line 4 and 7–10.

(b) We show how to solve the above recurrence using the tree methods presented in the course.

master method. The simplest way to solve the above recurrence is by using the master theorem. The recurrence is in the form $aT(n/b)+f(n)$ where $a = b = 2$ and $f(n) = c_2$. Note that $f(n) = c_2 = O(n^{\log_b a - \epsilon}) = O(n^{1-\epsilon})$ for any $0 < \epsilon \leq 1$. The recurrence falls in the first case of the master method, hence $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

recursion tree. Note that under the assumption that $n = 2^k$, the recursion tree describing the unraveling of the above recurrence is a full binary tree having internal nodes labelled with c_2 and leaves labelled with c_1. Since the number of leaves is equal to the number of internal nodes minus 1, we have that

$$T(n) = c_1 n + c_2(n-1) = (c_1 + c_2)n - c_2 = \Theta(n) \tag{1}$$

substitution method. Equation (1) can be proved by induction on k where $n = 2^k$. Equivalently, we show that for $k \in \mathbb{N}$, $T(2^k) = c_1 2^k + c_2(2^k - 1)$. Let us rewrite the recurrence in terms of k

$$T(2^k) = \begin{cases} c_1 & \text{if } k = 0, \\ 2T(2^{k-1}) + c_2 & \text{if } k > 0. \end{cases}$$

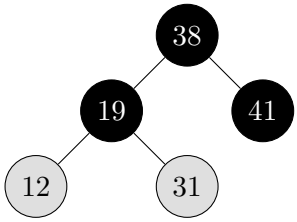
Base Case ($k = 0$). Then, $T(2^k) = c_1 = c_1 2^k + c_2(2^k - 1)$.
Inductive Step ($k > 0$). Then,

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + c_2 && \text{(by def. of } T\text{)} \\ &= 2(c_1 2^{k-1} + c_2(2^{k-1} - 1)) + b && \text{(by inductive hypothesis)} \\ &= c_1 2^k + c_2 2^k - c_2 \\ &= c_1 2^k + c_2(2^k - 1)\,. \end{aligned}$$

Question 3. 20 Pts

Understanding of known algorithms

(3.1) [5 Pts] Mark **ALL** the correct statements. Consider the red-black tree T depicted here below.



Here we refer to the implementation of RB-INSERT as described in CLRS Chapter 13.

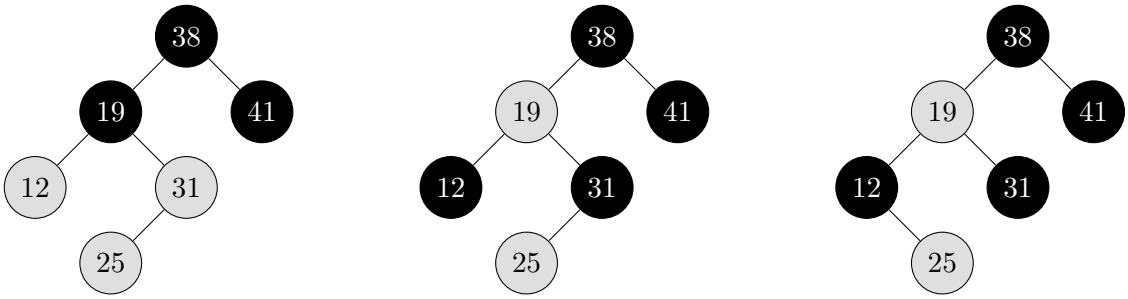
☐ **a)** The tree below to the left shows the result of RB-INSERT$(T, 25)$

☐ **b)** The tree below in the centre shows the result of RB-INSERT$(T, 25)$

☐ **c)** The tree below to the right shows the result of RB-INSERT$(T, 25)$

☐ **d)** None of the trees below represents the result of RB-INSERT$(T, 25)$

☐ **e)** Only one tree among the trees below satisfies the red-black tree property



(3.2) [5 Pts] Consider the hash table $H = \text{NIL}, 12, 101, 24, \text{NIL}, 5, \text{NIL}, \text{NIL}, 8, 20, \text{NIL}$. Insert the keys $18, 110, 49$ 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)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$ ☐ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$

☐ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$ ☐ **d)** none of the above

Mark the hash table resulting by the insertion of these keys using quadratic probing with $c_1 = 1$ and $c_2 = 4$.

☐ **a)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$ ☐ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$

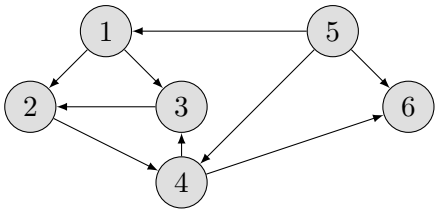
☐ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$ ☐ **d)** none of the above

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)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$ ☐ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$

☐ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$ ☐ **d)** none of the above

(3.3) [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. *Remark:* If more than one vertex can be chosen, choose the one with smallest label.

(b) Write the corresponding "parenthesization" of the vertices in the sense of Theorem 22.7 in CLRS

(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) Show the components computed by Strongly-Connected-Components(G).

Solution 3.

(3.1) The correct answers are (b) and (e).

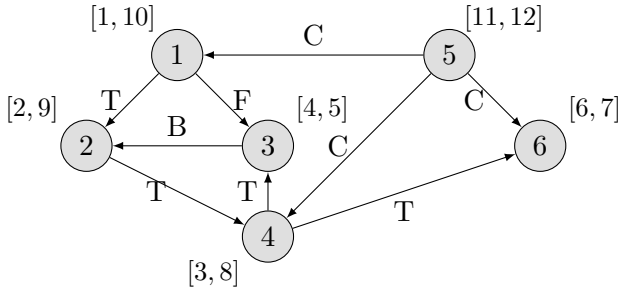
(3.2) The resulting hash tables are respectively:

linear probing: $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$

quadratic probing: $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$

double hashing: $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$

(3.3) 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

$$(1\ (2\ (4\ (3\ 3)\ (6\ 6)\ 4)\ 2)\ 1)\ (5\ 5).$$

(d) After running Strongly-Connected-Components(G) we obtain the following strongly connected components: $\big\{\{5\}, \{1\}, \{2, 3, 4\}, \{6\}\big\}$.

Question 4. 20 Pts

Peter works in Legoland where he is responsible of assembling buildings made of Lego blocks. He noticed that most of his work consists in repeating the following task: assemble a line of given length using Lego blocks. Given an unlimited supply of Lego blocks of given sizes, Peter wants to find the minimum amount of lego blocks required to form a line of length L.
For example, if each Lego block has one of following sizes $S = [3, 5, 7]$, the minimum amount of blocks required to form a line of length $L = 15$ is 3. Indeed, this can be achieved as $(5 + 5 + 5)$ or $(3 + 5 + 7)$. Sometimes, some lines cannot be formed using only the available blocks, e.g. there is no way to make a line of length $L = 4$.
Let's denote with $P(L, S)$ the minimum amount of blocks required to form a line of length L using Lego blocks of sizes in S. Then, $P(L, S)$ can be recursively defined as follows.

$$P(L,S) = \begin{cases} \infty & \text{if } L < 0 \\ 0 & \text{if } L = 0 \\ 1 + \min_{1 \le i \le S.length} P(L - S[i], S) & \text{if } L > 0 \end{cases}$$

Describe a bottom-up dynamic programming algorithm that computes $P(L, S)$ given $L \geq 0$ and an array S of block sizes. Analyse the worst-case running time of your algorithm.

Solution 4.

We can compute $P(L, S)$ in sequence as follows: first we compute $P(1, S)$, then $P(2, S)$, and so on until $P(L, S)$. The following procedure iteratively computes $P(i, S)$ for $i \in \{0, \ldots, L\}$ and returns the value $P(L, S)$. This is done by implementing the computation of the recursive characterisation for the function $P(L, S)$ in a bottom-up fashion, using an array $P[0\,..\,L]$ to to store the values $P[i] = P(i, S)$ for $i \in \{0, \ldots, L\}$.

```
BOTTOM-UP-BUILDLINE(L, S)
 1  create the array P[0..L]
 2  P[0] = 0
 3  for j = 1 to L
 4       // Compute min_{1≤i≤S.length} P(L − S[i], S)
 5       q = ∞
 6       for i = 1 to S.length
 7            if j − S[i] ≥ 0 and q > P[j − S[i]]
 8                 q = P[j − S[i]]
 9       P[j] = 1 + q
10  return P[L]
```

The overall running time of the above procedure is $\Theta(L \cdots |S|)$, since initialising P takes $\Theta(L)$, and the two nested for-loops overall take $\Theta(L \cdot |S|)$. This suffices to solve the question with full marks.
For those interested on how one can actually retrieve the optimal list of bricks, here is how you can do that by simply using another array to store the optimal choices. We modify the above solution to keep track of optimal brick choices in the array $B[1\,..\,L]$ as follows (in blue are the parts that differ from the previous pseudocode above)

```
BOTTOM-UP-BUILDLINE(L, S)
 1  create the arrays P[0..L] and B[1..L]
 2  P[0] = 0
 3  for j = 1 to L
 4       // Compute min_{1≤i≤S.length} P(L − S[i], S)
 5       q = ∞
 6       for i = 1 to S.length
 7            if j − S[i] ≥ 0 and q > P[j − S[i]]
 8                 q = P[j − S[i]]
 9                 B[j] = i
10       P[j] = 1 + q
11  return P[L] and B
```

Now, one use the array of choices B and print an optimal list of bricks forming a line of length L as follows.

```
PRINTBLOCKS(L, S)
1  (v, B) = BOTTOM-UP-BUILDLINE(L, S)
2  if v < ∞ // Check if there is a solution
3       j = L // Initialise length of the subproblem
4       while j ≠ 0
5            print S[B[j]] // Print chosen brick
6            j = j − S[B[j]] // Update length of the subproblem
```

The worst-case running time of the PRINTBLOCKS algorithm is $\Theta(L \cdot |S|)$: the call to the subroutine BOTTOM-UP-BUILDLINE(L, S) takes $\Theta(L \cdot |S|)$ and the **while**-loop takes $\Theta(L)$.

Question 5. 20 Pts

Consider a weighted directed graph $G = (V, E)$ representing the map of a city where the weight of each edge $(v_i, v_j) \in E$, namely $w(v_i, v_j)$, represents the distance from the location v_i to v_j.

(5.1) [10 Pts] Assume that the vertices $H \subset V$ represent the hospitals in the city. Describe an algorithm which assigns each vertex $v \in V$ to the nearest hospital, that is, the hospital $h \in H$ such that the distance from h to v is minimal. Analyse the running time of your algorithm.

(5.2) [10 Pts] Assume now that we have to choose where to place the fire station in the city. Describe an algorithm which determines a vertex $f \in V$ which minimises the mean distance from all the other vertices, that is $\min_{f \in V} \sum_{v \in V} \delta(f, v)/|V|$. Analyse the running time of your algorithm.

Solution 5.

(5.1) Note that by construction the weights are non negative, therefore we do not need to worry about negative cycles, and we can use Dijkstra's algorithm. For simplicity we assume that the vertices V are indexed from 1 to $|V| = n$, that is $V = \{v_1, \ldots, v_n\}$. The following algorithm takes a directed weighted graph (G, w) and a set of vertices $H \subset V$ and returns an array $B[1\,..\,n]$ such that $B[j] = i$ if and only if the vertex $v_j \in V$ is assigned to the hospital $v_i \in H$.

```
AssignHospital(G, w, H)
 1  let B[1..n] ba an array initialised with 0s // Current hospital assignments
 2  let A[1..n] be an array initialised with ∞s // Distance from current hospital
 3  for each v_i ∈ H
 4       Dijkstra(G, w, v_i) // Compute distances from v_i
 5       // Update current assignments
 6       for each v_j ∈ V
 7            if A[j] > v_j.d
 8                 A[j] = v_j.d
 9                 B[j] = i
10  return B
```

The running time of Dijkstra's algorithm depends on the data structure used for implementing the min-priority queue. In class we have seen that using a simple array implementation, Dijkstra's algorithm runs in $O(|V|^2)$. Therefore the overall running time of AssignHospital(G, w, H) is $O(|H| \cdot |V|^2)$.

(5.2) Again, we assume that the vertices V are indexed from 1 to $|V| = n$, that is $V = \{v_1, \ldots, v_n\}$. The following algorithm takes as input an $n \times n$ matrix W defined as in Equation (25.1) of CLRS, and returns an index $1 \leq f \leq n$ corresponding to the

```
FireStation(W)
 1  n = W.rows // number of vertices
 2  m = ∞ // current mean
 3  f = 0 // current fire station index
 4  D = Floyd-Warshall(W) // compute all-pairs shortest-paths
 5  // Determine the least mean distance
 6  for i = 1 to n
 7       q = 0
 8       for j = 1 to n
 9            q = q + D[i, j]
10       if q < m
11            // Update current mean and fire station
12            m = q
13            f = i
14  return f
```

The running time of FireStation(W) is $\Theta(|V|^3)$ since the initialisation of m, f and n takes $\Theta(1)$, the call Floyd-Warshall(W) takes $\Theta(|V|^3)$ and the overall cost of the outer for-loop is $\Theta(|V|^2)$.

Alternatively, one can solve the problem by repeatedly solving a single-source shortest path problem, e.g., using Dijkstra's algorithm. The following algorithm takes as input a directed weighted graph (G, w) and returns a vertex that minimises the average distance from all other vertices.

```
FireStation(G, w)
 1  m = ∞ // current mean
 2  f = NIL // current fire station
 3  // Determine the least mean distance
 4  for each s ∈ V
 5       Dijkstra(G, w, s)
 6       q = 0
 7       for each v ∈ V
 8            q = q + v.d
 9       if q < m
10            // Update current mean and fire station
11            m = q
12            f = s
13  return f
```

The running time of FireStation(W) is $O(|V|^3)$ since the initialisation of m and f takes $\Theta(1)$, each call to Dijkstra(G, w, s) takes $O(|V|^2)$ and the inner-loop takes $\Theta(|V|)$.

It is worth noting that the running time of the second solution is not tight, in contrast to the first proposed solution which was using the Floyd-Warshall algorithm.