Exam - June 2021

ALG-CPH-F21: Algorithms and Data Structures (SW2)

Instructions. This exam consists of **four questions** and you have **four hours** to solve them. Note that each question begins on a new page. The maximum possible score is **100 points**. You can answer directly on this paper, use additional sheets of paper, or write directly into a document on your computer.
Remember to upload your answers to the exam as a single PDF file. After creating the PDF file, please verify that you are able to open it on your computer before submitting, and that any scans/pictures you have taken are clear.
One option is to write your responses directly in, e.g., Google Docs, and export a PDF to hand in. Make sure to clearly indicate which question you are answering, and how. For instance, for a multiple choice question, you might indicate your answers in this way:
"Question 0.0: a) TRUE, b) FALSE, c) TRUE, d) FALSE" You do not need to copy the question text, as long as it is clear what you are responding to.

- Before starting with the exam, please make sure you read and understand these guidelines.
- For each question in the exam, make sure you read the text carefully, and understand what the problem is before solving it. Terms in bold are of particular importance.
- During the exam, you are allowed to use: a calculator, an overview of logarithm rules, **CLRS**, any course-notes you may have, all content available on the ALG KBH F21 Moodle page[1], including all slides, exercises, previous exams, and solutions to all of these.
- **CLRS** refers to the textbook we have used in the course: T.H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition).
- You are **not** allowed to communicate with others during the exam, beyond asking for administrative help from the administrative staff.
- If you prefer to answer on physical paper, please make an effort to use readable handwriting, and make sure you have a good way of scanning your answers before beginning (e.g. a scanner, high-quality camera, etc.).
- It is highly recommended to have a look at the **entire** exam before starting, so you get an idea of the time each question might take.
- You may answer in English, Danish, or any other Scandinavian language.

[1] `https://www.moodle.aau.dk/course/view.php?id=37435`

Question 1. 20 Pts

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

(1.1) [5 Pts] Mark **ALL** the correct answers. $3000 + \lg n^2 + \sqrt{n^2} + \lg 8^n$ is:

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

(1.2) [5 Pts] Mark **ALL** the correct answers. $40! + n + n \lg n$ is:

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

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

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

(1.4) [5 Pts] Mark **ALL** the correct answers. Consider the following recurrence

$$T(n) = \begin{cases} 200 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

☐ **a)** The first case of the master theorem can be used to prove that $T(n) = \Theta(n^2)$.

☐ **b)** The master method can be used to solve this recurrence.

☐ **c)** The substitution method can be used to solve this recurrence.

☐ **d)** Using the master method, it is possible to prove that $T(n) = \Theta(n \lg n)$.

☐ **e)** Using third case of the master theorem, it is possible to prove that $T(n) = \Theta(\sqrt{n})$.

Solution 1.

(1.1) $= n = \Theta(n)$. Correct answer: (b).

(1.2) $= n \lg n = \Theta(n \lg n)$. Correct answers: (c, d, e).

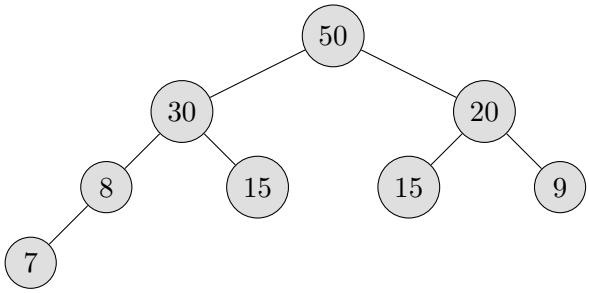
(1.3) $= 5^n$ Correct answers: (a, b, c, d, e).

(1.4) Master method, using $a = b = 2$ and $f(n) = n$. Yields $n^{\lg_2 2} = n$. Using case 2, since $f(n) = \Theta(n)$. Hence, by case 2, $T(n) = \Theta(n^{\lg_a b} \lg n)) = \Theta(n \lg n)$.

Correct answers: (b, c, d).

Question 2. 30 Pts

(2.1) [4 Pts] Mark **ALL** the correct statements. Consider the tree T depicted below



- ☐ **a)** T satisfies the Max-Heap property
- ☐ **b)** The height of T is 3
- ☐ **c)** T satisfies the binary search tree property
- ☐ **d)** T corresponds to a binary tree interpretation of the array $[50, 30, 20, 8, 15, 15, 9, 7]$

(2.2) [4 Pts] Mark **ALL** the correct statements. Consider a modification to QUICK-SORT, which first randomises the input-array before sorting. We will call this procedure RANDOMIZE-AND-THEN-QUICK-SORT, or RATQ-SORT for short. Assume that randomisation takes constant time.

- ☐ **a)** RATQ-SORT will in practice **some times** have a faster running time than QUICK-SORT.
- ☐ **b)** RATQ-SORT has the same asymptotic time as QUICK-SORT
- ☐ **c)** RATQ-SORT will guarantee that we **always** avoid the worst-case running time of QUICK-SORT.
- ☐ **d)** QUICK-SORT uses $\Theta(1)$ **space**.

(2.3) [22 Pts] Consider the following recursive algorithm, BEST-MEME, which calls the subprocedure MEME-QUALITY on each element in the array. The input is a non-empty array $A[1 \ldots A.length]$ and two indices p and q such that $1 \leq p \leq q \leq A.length$. When the recursion bottoms out (lines 1-2), the procedure MEME-QUALITY is called. Assume that the MEME-QUALITY procedure **uses constant time**.

```
BEST-MEME(A, p, q)
 1  if q = p
 2      return MEME-QUALITY(A[p])
 3  else
 4      m = ⌊(p + q)/4⌋
 5      left = BEST-MEME(A, p, m)
 6      right = BEST-MEME(A, m + 1, q)
 7      if left < right
 8          return left
 9      else
10          return right
```

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 BEST-MEME.

(b) [6 Pts] We will now assume that the objective of BEST-MEME is to return the entry from A with the highest score from MEME-QUALITY. Under this assumption, there are two errors in BEST-MEME. Which two lines need to be updated to solve this error, and in which way, so that the entire input array is checked, and the entry with the highest value from MEME-QUALITY is returned? We will denote this updated version as ACTUAL-BEST-MEME.

(c) [3 Pts] Write the recurrence $T(n)$ describing the running time of ACTUAL-BEST-MEME.

(d) [10 Pts] Prove that the algorithm ACTUAL-BEST-MEME, given your recurrence from 2.3 (c), runs in in $\Theta(n)$.

Solution 2.

(2.1) The tree does satisfy the Max-Heap property, as all children are smaller than their parents. The height of the tree is 3 since its longest path from the root to a leaf has 3 edges. The tree does not satisfy the BST property, as, e.g., $20 > 50$. The tree does correspond to the binary tree interpretation of the array.

Correct answers: (a, b, d).

(2.2) There are cases where RATQ-SORT may be faster, e.g., if the original input is reversed and the randomised is not. However, asymptotically this will be the same, as RATQ-SORT $= \Theta(\text{QUICK-SORT}) + c = \Theta(\text{QUICK-SORT})$. There is no guarantee, however, that this will avoid the worst-case running time. Finally, quick sort does not use constant space.

Correct answers: (a, b).

(2.3) (a) The recurrence should have the form

$$T(n) = \begin{cases} c_1 & \text{if } n \leq 1, \\ 2T(n/4) + c_2 & \text{if } n > 1. \end{cases}$$

where c_1 is a positive constant representing the running time for the first few lines, and c_2 is another positive constant representing the aggregated running time for the last few lines.

(b) There are two errors here. One is that we divide by 4 on line 4. This should be a division by 2. Furthermore, the comparison on line 7 is wrong, and should be reversed to check that right is greater than left.

(c) This bugfix yields the recurrence:

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

(d) The easiest way to do this is to use the master theorem, case 1. This is possible since $f(n) = O(n^{\log_b a)}$, as $f(n) = \Theta(1)$ and $O(n^{\log_b a)} = O(n)$, yielding $T(n) = \Theta(n^{\log_b a)} = \Theta(n)$. A proof via the substitution method or a recurrence tree would also be acceptable.

Question 3. 25 Pts

You are a spy working behind enemy lines, and have been given the task of decoding enemy messages. The enemy uses a simple encryption consisting of simply removing all white space from their messages. Luckily, you (1) have access to a dictionary of possible words in the enemy language (E), and (2) are a **software spy** with access to algorithmic tools.
You are given an encrypted input sequence s, and you need to print a list of possible output sequences. For instance, given the dictionary $E = \{"there", "the", "are", "rear", "emails", "mails"\}$, and the input string $s = "therearemails"$, your algorithm should print (in any order):

- "there are mails"
- "the rear emails"

(3.1) [10 Pts] Describe a *recursive* algorithm which **prints** the output as described above. Use pseudocode for this, and name the procedure DECODE.

(3.2) [10 Pts] Consider a case where the enemy gives you an array of input strings, S, consisting of n **copies** of the original input s. Describe how you can use **dynamic programming** to improve the running time of an implementation of DECODE in this type of scenario.

(3.3) [5 Pts] If we assume that an implementation of DECODE from 3.1 finishes in $T(m)$ time, how much time would the dynamic programming solution from 3.2 take to process S, containing n copies of s?
Express your answer in terms of $T(m)$, e.g., *"$\lg T(m) \cdot s^2 + c$ where c is a constant factor"*

Solution 3.

(3.1) This is a reformulation of the Word Break Problem. A solution could look like this:

```
DECODE(E, s, out)
1  if s.length == 0
2        PRINT out
3
4  for i = 0 to s.length + 1
5        prefix = s[:i]
6        if prefix ∈ E
7              DECODE (E, s[i:], out + " " + prefix)
8
```

(3.2) Implementing Memoisation is the most straight-forward solution here.

(3.3) Still just m time, as the memoised implementation will only add a constant time factor. If a different dynamic programming solution was suggested in 3.2, a different and correct answer in 3.3 will naturally also be judged as correct.

Question 4. 25 Pts

Consider a weighted directed graph $G = (V, E)$ representing the map of a city where the weight of each edge $(v_i, v_j) \in E$ represents the distance from location v_i to v_j. The subset of vertices $B \subset V$ represents all bars in the city, and the remaining locations are denoted by $H = (V \setminus B)$. Assume that:

- The graph is complete, i.e., there exists an edge between each vertex $(v_i, v_j) \in V \times V$.
- The graph has negative weights, but no negative cycles.

(4.1) [15 Pts] Describe an algorithm that computes the **length** of the shortest route between all pairs of vertices $(v_i, v_j) \in H \times H$, with the requirement that all routes must go via **exactly one** of the bars in B. I.e., the shortest path from v_i to v_j with exactly **one** intermediate vertex in B. Analyse the running time of your algorithm.

(4.2) [10 Pts] You decide to go on a bar-crawl. Describe an algorithm which computes and returns a **route** (e.g. as an array A containing the order of vertices), which allows you to visit **all** of the bars in B **exactly once**. Note that the route via **all** bars does **not** need to be the shortest possible route. However, it should be a shorter route than simply going to each bar in the ordering denoted by the vertex indexation (we assume this order to be random). Suggest at least one heuristic which should give you a shorter path than this random ordering. Analyse the running time of your algorithm.

Solution 4.

(4.1) This is almost exactly the same as an exercise from the exercise session ("Rick and Morty"). One solution is to run Floyd-Warshall first (on line 1), and then continue as described:

```
OneBar(H, B)
1  D = Floyd-Warshall(H)
2  D' = ∞ matrix
3  for each v_i ∈ H
4        for each v_j ∈ H
5              for each b ∈ B
6                    d'_ij = min(d'_ij, d_ik + d_kj)
7
8  return D'
```

This is $\Theta(|V|^3)$

(4.2) This one gives some room for interpretation. The simplest would be a linear solution by vertex ordering (which would not give points, as indicated in the text), such as:

```
BarCrawl(B)
1  let A be an array representing our path
2  for each v_i ∈ B
3        A[i] = v_i
4  return A
```

A different straight-forward / greedy solution, would be

```
BarCrawl(B)
 1  let A be an array representing our path
 2  let i = 0
 3  A[i] = B[i]
 4  while B ≠ ∅
 5        for each v_j ∈ B
 6              if d(v_i, v_j) < d(v_i, A[i + 1])
 7                    A[i + 1] = v_j
 8        B = B ∖ A[i + 1]
 9        i = i + 1
10  return A
```

Which should be $\Theta(|B|^2)$.

Thank you for your hard work during this course!
Remember to upload your answers to the exam as a single PDF file. After creating the PDF file, please verify that you are able to open it on your computer before submitting. Once you are done and have handed the exam in, I would appreciate it if you could complete this short Google Form.[2] :)

Best of luck!
- Johannes

[2] `https://tinyurl.com/alg-cph-f21`