

Exam - June 2022

ALG-CPH-F22: 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**, but bear in mind that these points are only advisory, and will be weighted following the exam. Please provide your answers digitally, either in a separate document, directly on this exam PDF, or both.

Remember to upload your digital answers as a single PDF file. It will not be possible to hand in anything physically for this exam. Throughout the exam, make sure to clearly indicate which question you are answering.

- 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 **all** aides, including notes, slides, and your internet access / online search to the best of your abilities. However, you are **not** allowed to use tools which **directly solve** the problem you are working on. For instance, using a Master Theorem solver is not allowed.
- You are **not** allowed to communicate with others during the exam, beyond asking for administrative help from the administrative staff.
- 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, Norwegian, Swedish, or in any combination of these.

Question 1.

20 Pts

Identifying asymptotic notation. (Note: \lg means logarithm in base 2)(1.1) [5 Pts] Mark **ALL** the correct answers. $3 \lg n + 5\sqrt{n^4}$ 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. $5 \lg n^2 + \sqrt{n^2} + \lg n$ 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. $\sqrt{n^2} \lg n + 30n + 42^2 \lg n$ 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

$$T(n) = \begin{cases} 5 & \text{if } n \leq 1 \\ 5T(n/6) + n \lg \sqrt{n^2} & \text{if } n > 1 \end{cases}$$

- ☐ a) The master method can be used to solve this recurrence.
- ☐ b) The substitution method can be used to solve this recurrence.
- ☐ c) Using the first case of the master theorem, we can prove that $T(n) = \Theta(n^2)$.
- ☐ d) Using the second case of the master theorem, we can prove that $T(n) = \Theta(n)$.
- ☐ e) Using the third case of the master theorem, we can prove that $T(n) = \Theta(n \lg n)$.

Solution 1.(1.1) $= n^2 = \Theta(n^2)$. Correct answer: (d).(1.2) $= n = \Theta(n)$. Correct answers: (b, c, d, e).(1.3) $= n \lg n$ Correct answers: (a, b, c).(1.4) Master method, using $a = 5$, $b = 6$ and $f(n) = n \lg n$. Using case 3, since $f(n) = \Omega(n)$. Hence, $T = \Theta(n \lg n)$.

Correct answers: (a, b, e).

Linear Probing: $h(k, i) = (h'(k) + i) \bmod m$

Quadratic Probing: $h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$

Double Hashing: $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$

- (2.1) [9 Pts] Consider inserting the keys 20; 11; 41; 23; 5; 6; into a hash table of length $m = 8$, represented by the 8 boxes by each probing function. Write down the result of inserting these keys with the following hash functions, bearing in mind the probe definitions listed above. Assume indexation starting at 0, and enter the numbers into the corresponding boxes.

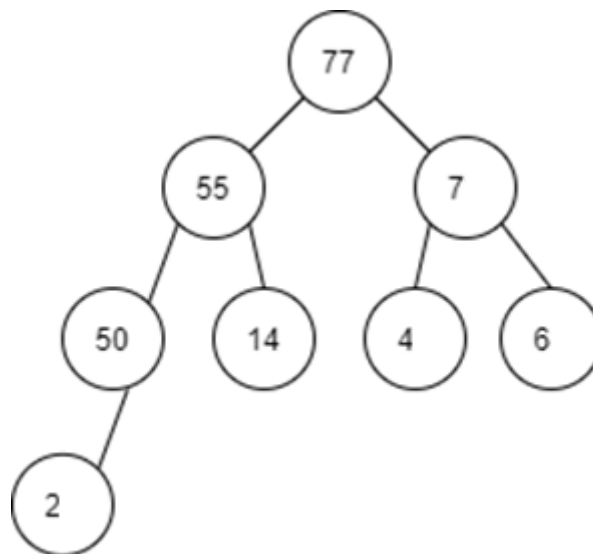
Index: 0 1 2 3 4 5 6 7

a) **Linear Probing** with $h'(k) = k$: ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

b) **Quadratic Probing** with $h'(k) = k, c_1 = 2, c_2 = 3$: ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

b) **Double Hashing** with $h_1(k) = k, h_2(k) = (2k)$: ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

- (2.2) [5 Pts] Consider the following tree, T , and mark **ALL** the correct answers below:



- ☐ a) T satisfies the Max-Heap property
- ☐ b) T satisfies the binary search tree property
- ☐ c) If all nodes in the tree are coloured *red*, with the exception of the root which is coloured *black*, T satisfies the Red-Black Tree property.
- ☐ d) The height of T is 3
- ☐ e) T corresponds to a binary tree interpretation of the array [77, 55, 7, 50, 14, 4, 6, 2]

- (2.3) [12 Pts] Consider the following recursive algorithm, MOST-CATS. The input is an array $A[1 \dots A.length]$ of pictures, some of which include one or more cats. When the recursion bottoms out, the procedure calls a cat-detection procedure, DETECT-N-CATS, which returns the number of cats present in the picture. Assume that DETECT-N-CATS uses constant time.

```

MOST-CATS( $A$ )
1  if  $length(A) == 1$ 
2      return DETECT-N-CATS( $A[1]$ )
3  else
4       $newsize = length(A)/4$ 
5       $a = \text{MOST-CATS}(A[0 : newsize])$ 
6       $b = \text{MOST-CATS}(A[newsize : 2 * newsize])$ 
7       $c = \text{MOST-CATS}(A[2 * newsize : 3 * newsize])$ 
8       $d = \text{MOST-CATS}(A[3 * newsize : 4 * newsize])$ 
9      return  $max(a, b, c, d)$ 

```

Assume that the input size at the first call is $n = 4^k$ for some $k \in \mathbb{N}$. Further assume that the notation $A[i:j]$ returns a sub-array of A from index i (inclusive) to index j (exclusive).

- (a) [4 Pts] Write the recurrence $T(n)$ describing the running time of MOST-CATS.
 - (b) [8 Pts] Use the Master Theorem to prove the running time of this recurrence, showing all the steps you use to reach your conclusion.
- (2.4) [10 Pts] In this assignment, you will work with a Doubly Linked List. Assume that your list has an attribute `L.head` which points at the head of the list (*nil* if `L` is empty). Further assume that each element in the list has the attributes `x.next` (*nil* if `x` is the tail) and `x.prev` (*nil* if `x` is `L.head`).
- (a) [7 Pts] Give a procedure which takes as input a Doubly Linked List, reverses it, and returns the reversed list. The procedure **must** run in linear time, i.e., $\Theta(n)$. Name the procedure REVERSE-DOUBLY-LINKED-LIST. Write pseudocode and provide a brief description in words how your procedure works.
 - (b) [3 Pts] Analyse the runtime of your version of REVERSE-DOUBLY-LINKED-LIST.

Solution 2.

- (2.1) There are no collisions, hence all probing mechanisms will return the same sequence: $_, 41, _, 11, 20, 5, 6, 23$
- (2.2) 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 correspond to the binary tree interpretation of the array.

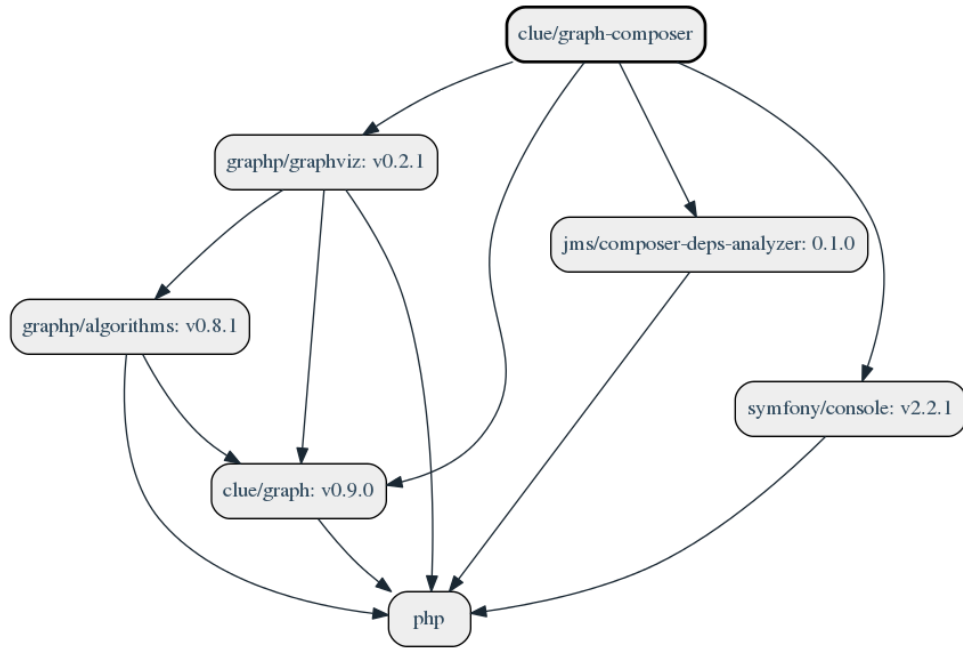
Correct answers: (a, d, e) .

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

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

- (b) This should evaluate to $\Theta(n)$, which intuitively is clear as each element in the array is checked once. Using the Master Theorem, the same conclusion can be reached, as $n^{\log_4 4} \geq f(n) = 1$ matches case 1.

- (2.4) (a) The solution must take into account an update of `L.head`, `x.next`, `x.prev`, and correctly use the properties of a doubly linked list. E.g. assuming that the input is an array, or that the length of `L` is known, or that `L` can be indexed as an array, is not a correct interpretation of the assignment. A single loop should suffice, starting at `L.head`, terminating when `x.next=nil`, and swapping each `x.next` with `x.prev`.
- (b) The following analysis should therefore also be relatively straight-forward in showing $\text{runtime} = \Theta(n)$



DFS(G)

```

1 for each vertex  $u \in G.V$ 
2    $u.color = WHITE$ 
3    $u.\pi = NIL$ 
4    $time = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.color == WHITE$ 
7     DFS-VISIT( $G, u$ )
  
```

TOPOLOGICAL-SORT(G)

```

1 call DFS( $G$ ) to compute finishing times  $v.f$  for each vertex  $v$ 
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices
  
```

DFS-VISIT(G, u)

```

1  $time = time + 1$ 
2  $u.d = time$ 
3  $u.color = GRAY$ 
4 for each  $v \in G.Adj[u]$ 
5   if  $v.color == WHITE$ 
6      $v.\pi = u$ 
7     DFS-VISIT( $G, v$ )
8  $u.color = BLACK$ 
9  $time = time + 1$ 
10  $u.f = time$ 
  
```

Question 3.

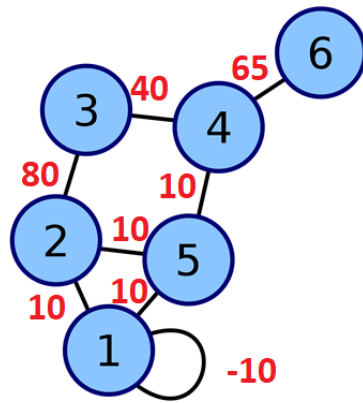
15 Pts

Consider the dependency graph for various software packages, G , as well as the pseudocode for Topological-Sort and DFS (all shown above).

- (3.1) [10 Pts] In order to compile this graph, we need to first perform a topological sort, in order to get a compilation order. Choose a starting vertex, and manually go through the steps of Topological-Sort on G . Provide **both** the resulting linked-list, as well as the **start** and **finish** times for each vertex during the DFS.
- (3.2) [5 Pts] Imagine a new case, in which there is a second graph, H , containing a distinct set of packages and dependencies, which is completely independent from G with a single exception: The node *php* in graph G has an outgoing arrow to a node in H , which **only** has outgoing arrows. If we now want to compile H , we want to take advantage of the fact that we already have a linked-list showing the compilation order for G . Describe in words how you can get a linked list with the compilation order of H independently from G . Which linked-list of packages should be compiled first, the one for G or the one for H ?

Solution 3.

- (3.1) A correct answer will contain a correct topological sort given as a linked list, and the corresponding start/finish times for the DFS.
- (3.2) Simply doing a topological sort on H without G would give a new linked list for compiling H. There is a node in H which depends on php, hence G must be compiled first. Then the topological sort of H can be followed.



```

BELLMAN-FORD( $G, w, s$ )
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i = 1$  to  $|G.V| - 1$ 
3      for each edge  $(u, v) \in G.E$ 
4          RELAX( $u, v, w$ )
RELAX( $u, v, w$ )
1  if  $v.d > u.d + w(u, v)$ 
2       $v.d = u.d + w(u, v)$ 
3       $v.\pi = u$ 

```

Figure 1: A graph G , and pseudocode for the procedures BELLMAN-FORD and RELAX

Question 4.

29 Pts

You are working with the implementation of route guidance for electric vehicles. Consider the above graph $G = (V, E)$, which represents possible routes within a small country. Each vertex V represents a city, and each edge E represents a road. The weights given by the weight function, $w : E \rightarrow \mathbb{R}$, representing the **battery consumption in percentage points** for driving from a city to the next. When it is possible to charge a vehicle's battery, this is currently indicated by the presence of a negative cycle in the graph (i.e. from vertex 1 to 1). Assume that your vehicle starts off with 100% battery power.

- (4.1) [6 Pts] Under normal circumstances (i.e. finding the shortest route without any other restrictions), the single-source shortest paths algorithms covered in the course do not work in this graph. Explain briefly why (i) *Bellman-Ford*, (ii) *Dijkstra's Algorithm* and (iii) *Single-Source Shortest Path in DAG* would not work.
- (4.2) [3 Pts] Examine the graph. Following the restrictions above (i.e. battery must remain ≥ 0), is there a possible route from 6 to 2? How about from 6 to 3? Write down these routes.
- (4.3) [20 Pts] Describe how you can solve the problem of route guidance for electric vehicles in the graph G , with the battery usage outlined above. As a hint, possible solutions can involve modification of **Bellman-Ford**, **Relax**, and/or **the graph G** . Explain how your proposed solution works, and describe the steps it would take to find a possible path from 6 to 3 with your solution. Take care that your solution does not allow the battery go below 0, and does not get stuck in an infinite charging loop. If you modify existing pseudocode, provide your answer in pseudocode as well as an explanation in words. If you modify the graph G , draw the new graph, and explain in words how your solution works.

Solution 4.

- (4.1) (i) no negative cycles allowed, (ii) assumes non-negative weights, (iii), G is not a DAG
- (4.2) Yes, e.g. 6 - 4 - 5 - 2 and 6 - 4 - 5 - 1 (repeat) - 2 - 3.
- (4.3) Many possibilities exist, as covered in detail in the lecture and the corresponding case-work. One relatively straight-forward solution could be to modify the data structure such that (1) is a special node with some property denoting this. Then, **RELAX** can be modified to disallow visiting this node once the battery has been re-charged up to 100. This would allow, e.g. BF, to find a route going down to 1, using max. 10 recharge-cycles, and then be forced to leave again. Note that this would also require BF to be modified to use more than $G.V-1$ iterations.

Another option is to first try to attain a straight-forward route from u to v , and if this is not possible, force the route from u to 1 , and then rerun BF for a route from 1 to v .

Thank you for your hard work and active participation in this course!
Remember to upload your answers to the exam as a single PDF file. Once you are done and have handed the exam in, I would appreciate it if you could complete this short Google Form.¹ :)

Best of luck!
- Johannes

¹<https://tinyurl.com/alg-cph-f22>