Exam - June 2023

ALG-CPH-F23: 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 **almost 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 rely on AI (e.g. no use of ChatGPT), or that **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. 15 Pts

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

(1.1) [5 Pts] Mark **ALL** the correct answers. $2\sqrt{n^4} + n \lg n$ 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. $1000 + \lg n + n$ 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. $10^{10}\sqrt{n^2} + n! + 10$ is

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

Solution 1.

(1.1) $= n^2$. Correct answer: (d).

(1.2) $= n$. Correct answers: (b, c, d, e).

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

Question 2. 10 Pts

Solve the following recurrence using the Master Method. Make sure that you show your calculations each step of the way, including which case of the Master Theorem the recurrence matches, and what the asymptotic analysis of the recurrence is. Make sure to simplify your final analysis as much as possible.

$$T(n) = \begin{cases} 42 & \text{if n} \leq 1 \\ 4\text{T}(4\text{n}/8) + \sqrt{n^4} & \text{if n} > 1 \end{cases}$$

Solution 2.

Master method, using $a = 4$, $b = 8/4 = 2$ and $f(n) = n^2$. Using case 2, since $n^2 = \Theta(n^2)$. Hence, $T = \Theta(n^2 \lg n)$.

Question 3. 20 Pts

Understanding of existing algorithms

(3.1) [6 Pts] Consider the binary tree interpretation, T, resulting from the array $A = [50, 40, 60, 20, 45, 55, 65]$ and mark **ALL** the correct answers below:

- ☐ **a)** T satisfies the Max-Heap property.
- ☐ **b)** T satisfies the binary search tree property.
- ☐ **c)** The height of T is 3.

(3.2) [4 Pts] Consider the same array A. Show the resulting array after calling Max-Heapify(A, 100).

(3.3) [5 Pts] Mark **ALL** the correct answers below:

- ☐ **a)** Merge-Sort and Insertion-Sort have the same worst-case asymptotic running time.
- ☐ **b)** Merge-Sort works in place
- ☐ **c)** A sorting algorithm that randomly permutes an array, has a best-case linear running time.
- ☐ **d)** The height of a balanced binary tree is $\Theta(\lg n)$
- ☐ **e)** The maximum height of a binary tree is $O(n)$

(3.4) [5 Pts] Mark **ALL** the correct answers below:

- ☐ **a)** The adjacency list representation of a graph $G = (V, E)$ requires $\Theta(|E|)$ space.
- ☐ **b)** BFS solves the all-pairs shortest paths problem
- ☐ **c)** BFS solves the single-source shortest paths problem
- ☐ **d)** In dynamic programming, we try to make use of overlapping sub-problems to increase the efficiency of algorithms
- ☐ **e)** Memoisation is a type of dynamic programming

Solution 3.

1. (b) is the only correct option
2. This problem has an error, as the second argument to max-heapify is supposed to be the index. This was meant to be, and commonly understood as, Max-Heapifying the whole array, hence points should be awarded both in this understanding of the issue, and when pointing out the error. This error has been taken into account in the general grading.
3. c, d, e are true. b is incorrect, as a standard merge-sort implementation requires an auxiliary array.
4. c, d, e are correct.

Question 4. 20 Pts

Analysis of new algorithms

(4.1) [20 Pts] Consider the following algorithm, Connected-Cities. The input is a graph $G = (V, E)$, denoting a city. The algorithm attempts to calculate the number of cities which are connected to more than n other cities.

```
Connected-Cities($G, n$)
1  $connected$ = new empty Array
2  for $v \in G.V$
3      $count = 0$
4      for $u \in G.adj[v]$ :
5          increment $count$
6      if $count \geq n$ :
7          append $v$ to $connected$
8  return $connected$
```

(a) [10 Pts] Write the asymptotic running time for each line in Connected-Cities.

(b) [5 Pts] What is the asymptotic running time of the entire procedure Connected-Cities? Make sure to simplify your answer as far as possible.

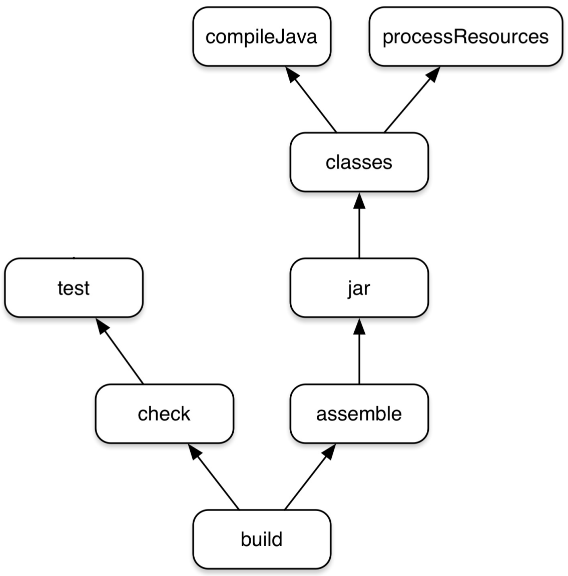
(c) [5 Pts] How does the asymptotic running time of the entire procedure Connected-Cities change if we add a condition to the for-loop on line 4, such that the loop breaks when $count >= n$? Write down the new asymptotic running time, simplified as far as possible.

Solution 4.

(4.1) Line 2: $|V|$, Line 4: $|E|$ - notice that this loops over all edges. The rest of the lines are constants.

(4.2) This sums to $\Theta(|V||E|)$

(4.3) This changes the loop running time to n. Hence the total time changes to $\Theta(|V|n)$



```
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  $u.f = time$
```

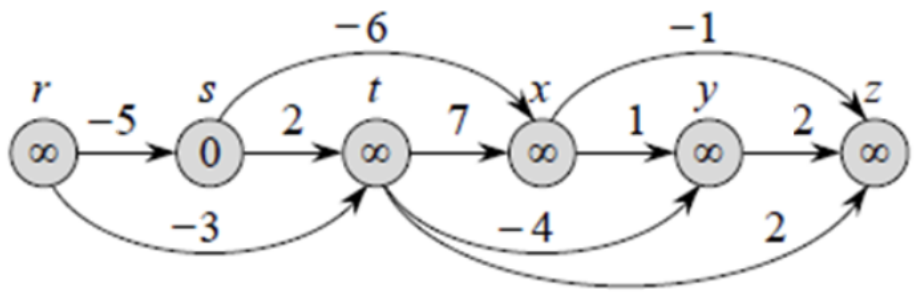
Question 5. 10 Pts

Consider the dependency graph for Java, 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 perform a topological sort. 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.

Solution 5.

(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.



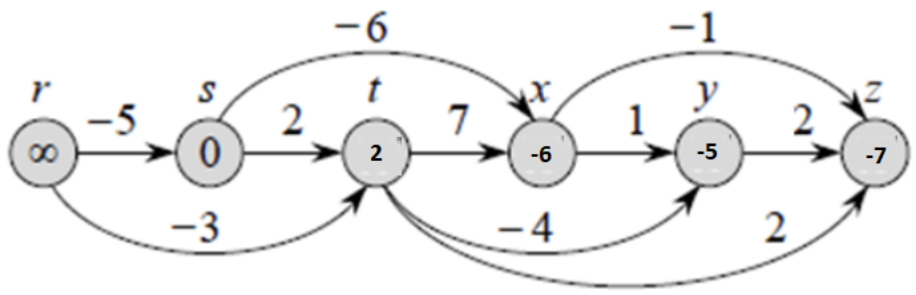
```
DAG-SHORTEST-PATHS$(G, w, s)$
1  topologically sort the vertices of $G$
2  INITIALIZE-SINGLE-SOURCE$(G, s)$
3  for each vertex $u$, taken in topologically sorted order
4      for each vertex $v \in G.Adj[u]$
5          RELAX$(u, v, w)$
```

```
RELAX$(u, v, w)$
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$
```

Question 6. 10 Pts

Consider the DAG above, showing the state of a graph after line 2 in DAG-SHORTEST-PATHS. Write down the resulting distances in the graph, after running the rest of the algorithm.

Solution 6.



Question 7. 15 Pts

You are given an unweighted graph $G = (V, E)$ representing movie actors, and their collaborations. $V = \{v_1, \dots, v_n\}$ is the set of all actors, and two actors are connected by an edge if they have acted in the same movie. Describe an algorithm that takes as input the graph $G = (V, E)$ and a subset of actors $A \subset V$. The algorithm should determine whether or not each actor $a_i \in A$ has acted together with at least one other actor $a_j \in A$. For instance, if $A = \{Dwight, Angela, Michael\}$, and Dwight and Angela have co-acted in one movie and Michael and Dwight have co-acted in one movie, the algorithm should return $True$.

(7.1) [10 Pts] Write pseudocode for your algorithm, and explain how it works in a couple of sentences. Your algorithm should assume an implementation of the graph using adjacency lists.

(7.2) [5 Pts] Provide an asymptotic analysis of your algorithm.

Solution 7.

(7.1) The simplest solution here is along the lines of:

```
ACTEDTOGETHER($G, A$)
let B denote a new empty set
for $v \in G.V$ :
    if $v \in A$ :
        for $costar \in A$ :
            if $costar \in G.adj[v]$
                add v to B
                break // Optional
if B = A
    return True
return false
```

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.

Best of luck!
- Johannes