Data Structures and Algorithms 2017/2018 Exam

First name, Surname: ROUGH KEY (DOUBLE CHECK)

Student ID: ______________________________

Program: Bachelor Data Science and Knowledge Engineering
Course code: KEN1420
Examiner: T.H.J. Pepels MSc.
Exam version: A
Date/time: Thursday April 5th, 2018, 9.00-12.00h
Format: Closed book exam
Allowed aides: Pens

Instructions to students:

- The exam consists of 6 questions on 23 pages (excluding the cover page).
- **Solve two problems out of A1, A2, and A3, and two problems out of B1, B2, and B3.**
- **If you do not follow the directions and solve all A problems, you will get credit only for A1 and A2. Similarly, if you solve all B problems, you will get credit only for B1 and B2.**
- Fill in your name and student ID number on each page, including the cover page.
- Answer every question at the reserved space below the questions. If you run out of space, continue on the back side, and if needed, use the extra blank pages.
- Ensure that you properly motivate your answers.
- Do not use red pens, and write in a readable way. Answers that cannot be read easily cannot be graded and may therefore lower your grade.
- You are not allowed to use electronic communication devices nor to wear or use a watch.
- You have to return all pages of the exam. You are not allowed to take any sheets, even blank, home.
- If you think a question is ambiguous, or even erroneous, and you cannot ask during the exam to clarify this, explain this in detail in the space reserved
- for the answer to the question.
- If you have not registered for the exam, your answers will not be graded, and thus handled as invalid.
- Good luck!

The following table will be filled by the examiner:

Questions:	**A1**	**A2**	**A3**	**B1**	**B2**	**B3**	**Total (of 4)**
Points:	20	20	20	20	20	20	80
Score:							

A1. (20 points)

1. (4 Points)
   The following algorithm takes as input an array, and returns the array with all the duplicate elements removed. For example, if the input array is:
   {1, 3, 3, 2, 4, 2}
   The algorithm returns:
   {1, 3, 2, 4}

```
S = new empty set
A = new empty dynamic array
for every element x in input array
  if not S.member(x) then
    S.insert(x)
      A.append(x)
return A
```

   What is the big-O complexity of this algorithm, if the set is implemented as:
   a. an AVL tree?

O(nlogn): For all elements (n), member+insert needs logn time.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

   b. a hash table

O(n): for all elements (n) assuming constant operations for the hash-table.

……………………………………………………………………………………………

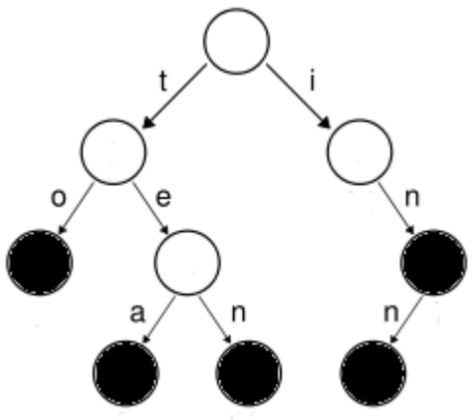
……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

(6 Points)

c. A trie is a kind of tree data structure used to store a sorted set of strings. To look up a string, begin at the root and follow the path of characters in the string. Nodes store a boolean value to indicate that the path from the root to that node represents a complete string stored in the set, and not a partial prefix. This allows a word that is a substring of another to be stored, for example, "in" and "inn" shown below. The edges in our trie will be labeled with just a single character.



Trie containing "to", "tea", "ten", "inn"

Assuming a 26-character alphabet, what is the Big-O complexity **of the lookup operation** for a trie containing n strings of maximum length m? How does this differ from a binary search tree containing the same set of strings?

Trie: We do n-lookups of max. length m, so for each operation is O(m)

Binary tree: We search for the string (there are n strings in total). If the tree is balanced, then it's O (logn)

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

d. Imagine you are implementing a spellcheck feature for a word processor. Your program needs to be able to identify misspelled words, and also propose corrections. Would a trie be a good data structure for storing a dictionary? Explain.

(different things you can say:

If you implement the spellcheck “on the fly”, i.e. as you write the words then the trie

is good, since you can immediately go back and check the “closest word”

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

2. (3 Points)
Mark the following statements true or false and give a short explanation or (counter) example.
   a. If computing something takes expected time $O(n^2)$ for an input of length n, there could be specific inputs for which the function returns in time $O(1)$

[✓ **T**] / F (if e.g. there is just one element)

…………………………………………………………………………………………

…………………………………………………………………………………………

   b. $O(3n^4 + 2n + 7)$ is the same as $O(n^4)$

[✓ **T**] / F…………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………………

c. Graphs: Using an adjacency list representation, you can determine if two vertices are connected by an edge in O(1) time.

T / **F** (in the general case where we have simply connected list, is O(n))

…………………………………………………………………………………………………

…………………………………………………………………………………………………

3. (7 Points)
Imagine that we are implementing a delete method for a binary search tree. Deleting a value in a leaf node with no children is easy, just “delete” the node by setting the parent’s reference to it to null.
However, deleting an internal node this way is not recommended as it will also disconnect its entire subtree.
Instead, we swap the value at the internal node to be deleted with that of a leaf node, and then delete the leaf. Which leaf would be a good candidate to swap the internal node’s value with so that the BST is still correct after the deletion and why?

(discussed in class as well)

Take the next leaf using the **inorder** traversal, since it will have the next biggest

value

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

A2. (20 Points)

1. (4 points)
   a. If all edge weights are equal, to what algorithm is Dijkstra's shortest paths algorithm equivalent?

BFS

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

   b. What property must a directed graph satisfy in order for topological sort to work on it?

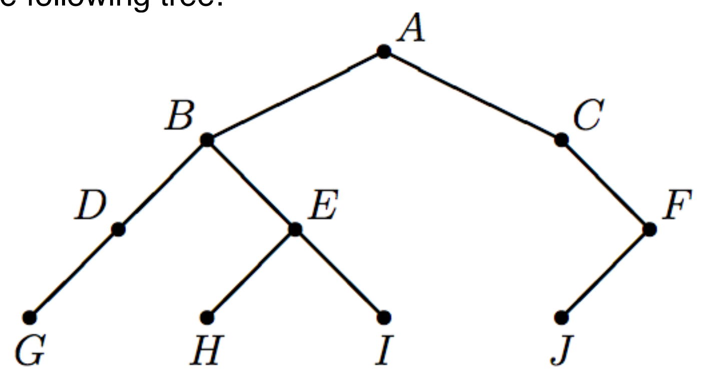
No cycles

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

2. (6 Points)
List the sequence of nodes visited by preorder, inorder, and postorder traversals of the following tree:



Preorder:

ABDGEHICFJ

……………………………………………………………………………………………

Inorder:

GDBHEIACJF

……………………………………………………………………………………………

Postorder:

GDHIEBJFCA

……………………………………………………………………………………………

3. (4 Points)
If we want a hash table that stores a set of strings, one possible hash function is the string's length, h(*x*) = *x.length*. Is this a good hash function? Explain your answer.

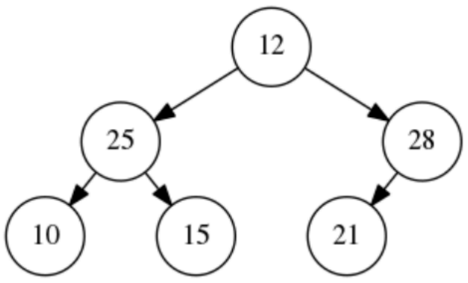
It's (very) bad. Length not a good idea (e.g. words like "cat", "dad" will map to the same entry)

……………………………………………………………………………………………

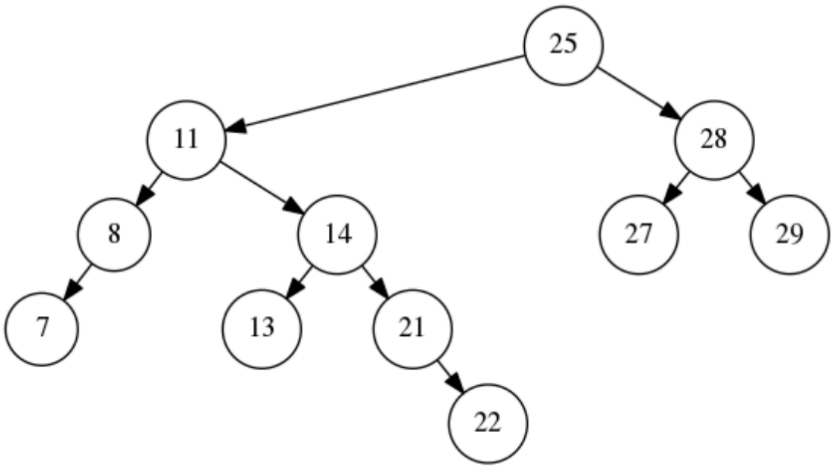
……………………………………………………………………………………………

4. (6 Points)
Have a look at the following three binary trees.

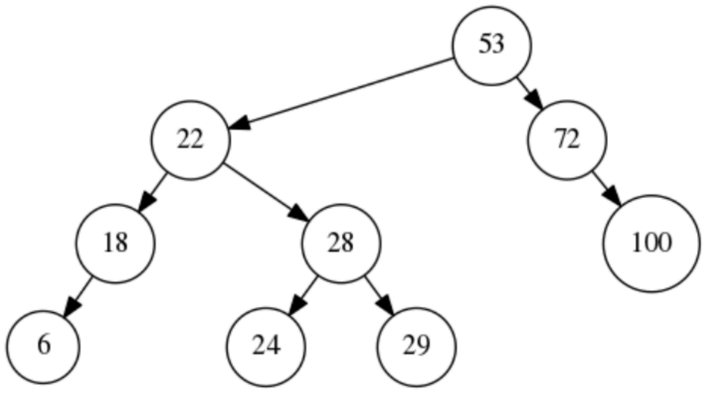
Tree A:



Tree B:



Tree C:



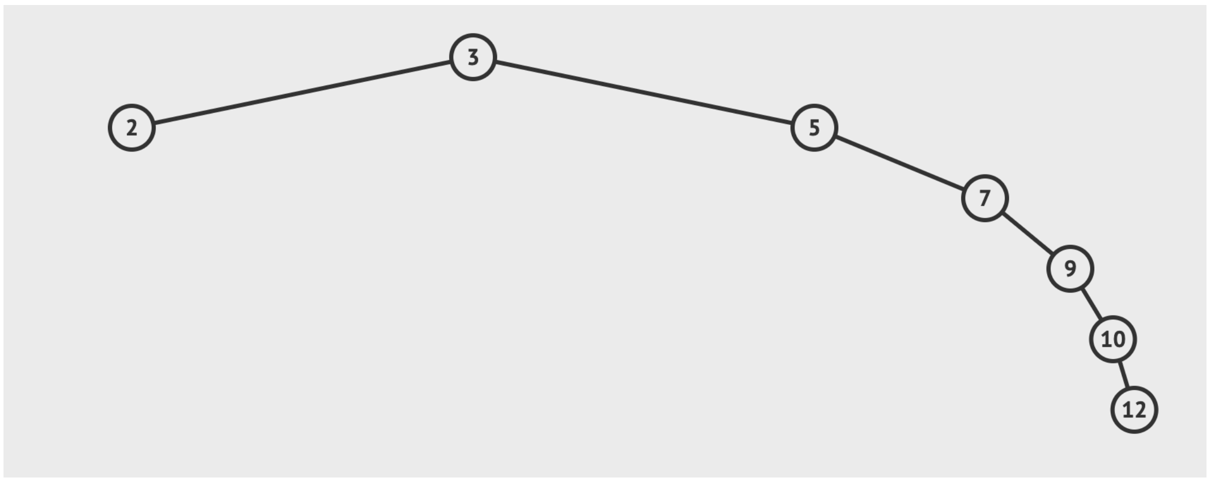
a) One of these trees is an AVL tree. Which one?

C

……………………………………………………………………………………………………

b) Draw the binary search tree that results after inserting these numbers (in order):
3, 5, 7, 2, 9, 10, 12.
It will help me give partial credit if you show the resulting tree after each insert.

(A3 - 4 b. Cont.)



A3. (20 Points)

1. (4 Points)
   The following questions concern the quicksort algorithm
   a. There are three fundamental steps to the Quicksort algorithm. What are they?

Divide, Recur and Conquer

(explain in detail):

Pick a pivot element, partition to 3 parts, apply quicksort to 1st/3rd

…………………………………………………………………………………………………

   b. Quicksort has an average case runtime of O(n log n), but a worst case complexity of O(n^2). Is this an issue in practice? Explain.

Bad pivot?

Already sorted?

All elements are the same?

2. (6 Points)
   Mark the following statements true or false and give a short explanation or (counter) example.
   a. A binary tree with at most 7 nodes has at most 3 levels.

T / **F**………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

   b. The **average** case complexity of every comparison based sorting algorithm is O(n log n), where n is the number of elements in the array.

T / **F**………………………………………………………………………………………

…………………………………………………………………………………………………

……………………………………………………………………………………………

c. If a graph *G* is connected, then the minimum spanning tree for *G* will also be connected.

T / F……………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

3. (5 Points)
   Give the asymptotic complexity in terms of n (e.g O(n^2)) and explain your answer. You should give the tightest possible big-Oh bound.

```
int sum= 0;
for (int i= 1; i <= n; i= i + 1) {
     for (int j= 1; j <= Math.min(1000000, i); j= j + 1) {
          sum= sum + 10;
     }
}

sum= sum + 9999;
```

O(n)

Outer loop: executed n times

Inner loop: constant for i>1000000

(if you mention that for i<1000000 it's O(n^2) it's also OK).

……………………………………………………………………………………………

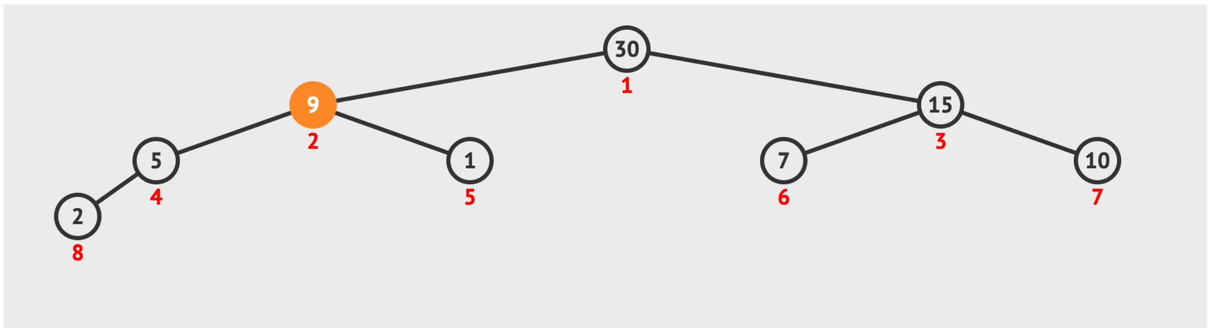
……………………………………………………………………………………………

……………………………………………………………………………………………

4. (5 Points)
   Draw the heap that results from inserting the following sequence of integers, in the order given, into a (min or max) heap.

   10, 5, 7, 2, 1, 30, 15, 9

   It will help me give partial credit if you show the heap after each insertion



(A3 – 4 Cont.)

B1. (20 Points)

Design an algorithm that takes two arrays, and returns true if the arrays are disjoint, i.e. have no elements in common.

You may freely use standard data structures and algorithms from the course in your solution, without explaining how they are implemented.

Write down your algorithm as pseudocode and in English. You don't need to write Java code, but be precise – a competent programmer should be able to take your description and easily implement it.

Show that your algorithm takes O(*n* log *n*) time.

Think: this is a problem that you could also solve with your CS1 knowledge (but then you would most probably do two loops, i.e. would need O(n^2) time.

Different ideas:
-Sort (any algorithm that needs O(nlogn)) and then you need O(n) for "merging"
-Sort (any algorithm that needs O(nlogn)) and then binary search for each element of the array to the other
-…

NB: If you use hashes and you take down time to O(n) that is still fine, since O(n)<O(nlogn)

(B1 cont.)

B2. (20 Points)

Design an algorithm that takes:

- An array containing n distinct natural numbers
- A number $k \leq n$ and calculates the sum of the k largest numbers in the array.

For example, if the array is {3, 7, 5, 12, 6} and $k = 3$, then the algorithm should return 25 (12+7+6).

You may freely use standard data structures and algorithms from the course in your solution, without explaining how they are implemented. Write down your algorithm as pseudocode and English.

Show that your algorithm takes O(n log n) time.

Easy (acc. To Tom ☺)

Sort the array (any algorithm that needs O(nlogn)) and then take the sum of the k (O(n))

(B2 cont.)

B3. (20 Points)

Consider a general tree *T*, which represents a file system. Thus, the internal nodes of *T* represent directories (or folders) and external nodes of *T* represent individual files. Assume further that each internal node uses 8 bytes of storage, and each external node has a size() method that returns the size of its associated file.

Describe in pseudo-code and in English a recursive procedure for computing the total size of all the files and directories in *T*. What is the running time complexity of your method in terms of *n*, the number of nodes in *T*?

Idea: Like DFS for trees, recursion necessary. Once you start a new recursive call (new "root") you add 8, and when you reach a leaf recursion ends and you return the size() of that leaf.

(B3 Cont.)

(Extra Answer Sheet)

(Extra Answer Sheet)

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

(Extra Answer Sheet)

(Extra Answer Sheet)

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………