DSA Exam Prep

Data Structures and Algorithms

Exam setup (closed books)

- Two parts (A and B), you can make choices;

- **A part** contains 3 problems (A1,A2,A3) short-answered, mostly-understanding problems. Answer two problems.
  - Study the slides and videos, read the book(s) for the sections you do not fully understand

- **B part** (B1, B2) contains algorithmic description problems. Answer one problem.

Exam Setup

- Read *all* 5 problems carefully 2 times to check:
  - If you understand the question
  - If you can provide a complete answer
  - Pick the **3 problems (2 from A, 1 from B)** that you think you know best how to approach

- Before attempting an answer, read the question again!
  - Sketch your answer, before writing it down
  - Especially for the **B** problems
  - Write an example input/output

Exam management

- Read the question very carefully
- Do not get stressed about time but do not lose track of time
- Reflect on (read and validate) your answers
- Do not panic! You are doing well so far 😊
- Study!

Please do not take this exam lightly, as we'd like to see you back with us again for future courses…
(it gets more fun throughout the years, we promise ;))

CodeGrade

Example Codegrade Assignment

Schoolyear Laptop Check

1. **Save your work!! All applications will close**
2. Go to: exams.schoolyear.app
3. Fill in code:

   T 8 N X 6 A

4. Follow on-screen instructions
5. Close the schoolyear environment, delete downloaded file.

The Example Exam!

Note: In the past, Exams at DKE lasted ***3 hours***. Hence, the example exams may be longer than the actual exam.

A1

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(n log n)

b. a hash table
O(n) (expected)
$O(n^2)$ (worst-case)

A1

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?

In case of a trie: ***O(m)***

For a binary search tree, each node stores a string, so when the tree is balanced we can perform search in ***O(log n)*** if not balanced ***O(n)***.

Note: binary search tree: means we have some way to order the strings

A1

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.

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 find the “closest word” in the try by determining where the spelling went wrong.

(See quiz week 6)

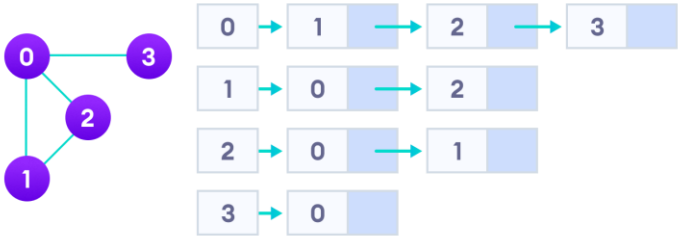
A1

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

True *(drop lower order and constant terms)*

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

False *O(n)*



A1

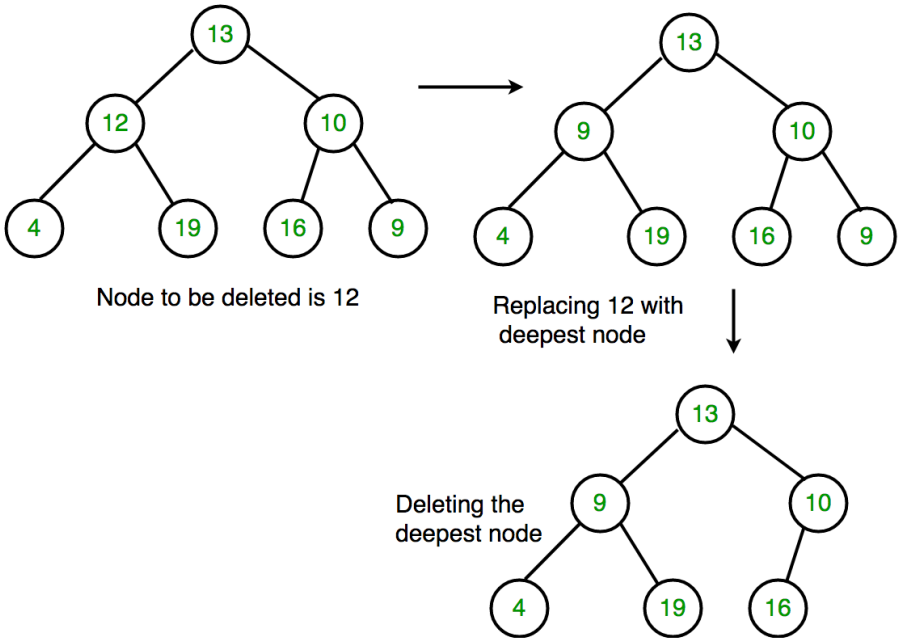
Imagine that we are implementing a delete method for a binary search tree.

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?

Take the last leaf using the inorder traversal, since it will have the next value



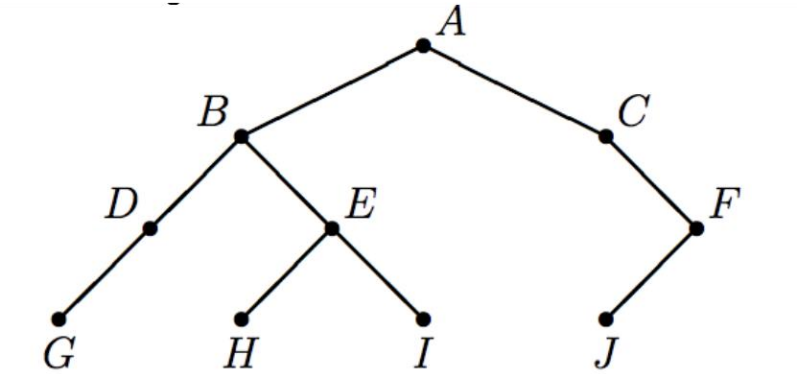
A2

If all edge weights are equal, to what algorithm is Dijkstra's shortest paths algorithm equivalent?
Breadth First Search

What property must a directed graph satisfy in order for topological sort to work on it?
It must have no cycles

A2

List the sequence of nodes visited by preorder, inorder, and postorder traversals of the following tree:



Preorder: ABDGEHICFJ
Inorder: GDBHEIACJF
Postorder: GDHIEBJFCA

A2

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.

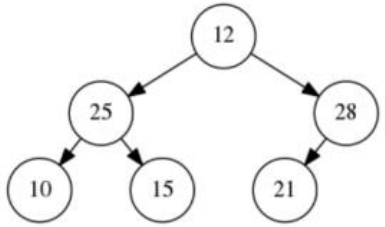
It's (very) bad.

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

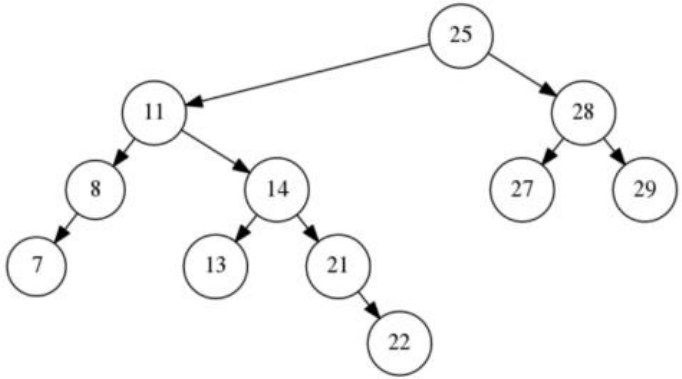
Many English words have the same length.

A2

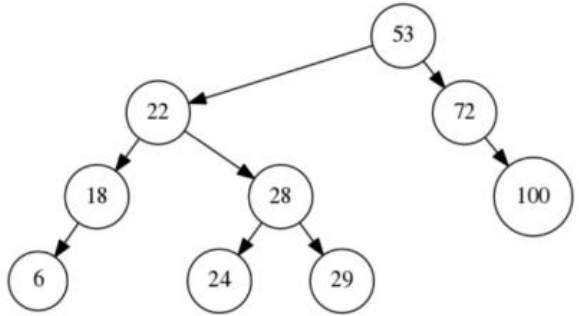
Tree A:



Tree B:



Tree C:



Which one is an AVL tree:

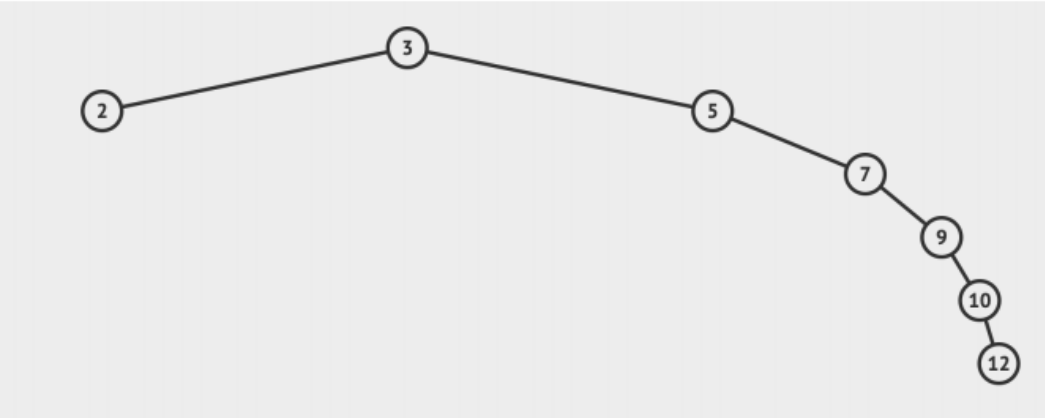
C

Because it is a balanced BST

A2

Draw the **binary search tree** that results after inserting these numbers (in order):

3, 5, 7, 2, 9, 10, 12



A3

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 parts and merge around the 2nd (the pivot)

Quicksort has an average case runtime of O(n log n), but a worst case complexity of O(n2).
Is this an issue in practice? Explain.

When a bad pivot is selected or for a specific input: ***1, 2, 2, 2, 2, 2, 2, 2, 2***

A3

A ***binary tree*** with at most **7** nodes has *at most* **3** levels:
False

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:
False

If a graph G is connected, then the minimum spanning tree for G will also be connected:
True

A3

Give the asymptotic complexity in terms of n (e.g. ***O(n²)***) 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;
```

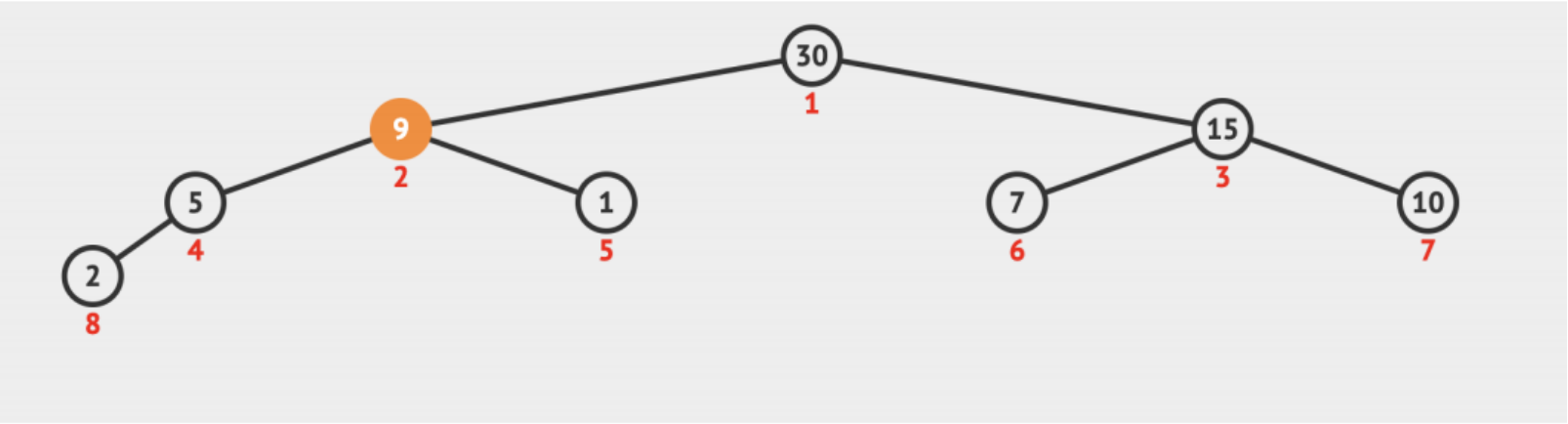
Answer: O(n)
Outer loop: executed ***n*** times
Inner loop: min(n, ***1.000.000***) i.e. constant

Remember: *Worst Case* and ***Growth*!** Are what matters.

A3

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



B1

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.

B1

Sort both arrays, loop over them and compare elementwise:

Ex: A [**4**, 5, 5, 6]

B [1, 2, 2, 3, **4**]

```
algorithm disjoint(A, B)
    sort(A), sort(B)                  2n log n
    j <- 0                               1
    for i from 0 to A.length             n
        if not A_i equals B_j then       n
            if A_i >= B_j                n
                j <- j + 1               n
        else return false                1
        if j >= B.length then break      1
    return true                          1
```

B2

Design an algorithm that takes:

- An array containing ***n*** distinct natural numbers
- A number ***k*** ≤ ***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 code and explain in english.

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

B2

Ex: A [16, 14, 5, 6 , 12]

```
algorithm maxk(A, k)
    sort(A)                                  n log n
    sum <- 0                                    1
    for i = A.length - k to A.length            n
        sum <- sum + A_i                        n
    return sum                                  1
```

B3

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***?

B3

```
algorithm fssize(node)
    if isleaf(node)
        return node.size()
    sum <- 8
    for each child of node
        sum <- sum + fssize(child)
    return sum
```

Given that we visit each node in the tree once, and at each node we run a constant number of operations (the number of operations run at each node does not depend on the size of the tree) we can conclude that the algorithm runs in **linear time; O(|T|)** in relation to the size of the tree T **(|T|)**

Thank you all!

Thank you for attending
Thank you for being our students
Thank you for doing our weird assignments
We enjoyed hosting this course for you!

Good luck with the exam!