Exercises

1	2	3	4	5	6

Surname, First name

_____

Data Structures and Algorithms (2122-KEN1420)
DSA Exam 2021/2022

Program: Bachelor Data Science and Artificial Intelligence
Course code: KEN1420
Examiners: F. Barile and T. Pepels
Date/time: Thursday 31 March 2022 / 13.00 - 15.00
Format: Closed book exam
Allowed aids: Pens, simple (non-programmable) calculator from the DKE-list of allowed calculators

Instructions to students:
- The exam consists of 5 questions on 34 pages.
- Fill in your name and student ID number on the cover page.
- Answer every question in the reserved space below the question. **Do <u>not</u> write outside the reserved space or on the back of pages, this will not be scanned and will NOT be graded!** As a last resort if you run out of space, use the extra answer space at the end of the exam.
- _In no circumstance write on the QR code at the bottom of the page!_
- Ensure that you properly motivate your answers.
- Before answering the questions, please first read all the exam questions, and then make a plan to spend the two hours.
- Only use black or dark blue pens, and write in a readable way. No pencils. Answers that cannot be read easily cannot be graded and may therefore lower your grade.
- 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.
- You are not allowed to have a communication device within your reach, 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.
- **Solve two problems out of A1, A2 and A3, and one problem out of B1 and B2**
- **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.**
- **The maximum total points is 60.**
- Good luck!

A1

This section has 6 questions with a maximum of 20 points in total.

A1.1: (10 points) Complexity / run time

5p **1a** Consider the algorithm represented by the following Java fragment. What value does the method **bar** compute? Derive a tight **big Oh** expression for the running time of the method **bar**. Explain how you arrive at this complexity in at least two, and at most five sentences

```java
private int bar(int x, int n) {
  int sum = 0;
  for (int i = 1; i <= n; ++i) {
    sum = sum + i;
  }
  return x + sum;
}
```

5p **1b** Consider the algorithm represented by the following Java program fragment. The method **bar1** computes the same method as bar in question 1a.
However, **bar1** is based on a different algorithm and its big oh bound is *O(n)*.
What value does **foo** compute? Give the tightest **big Oh** expression for the worst case running time of the method **foo**.

Explain how you arrive at this complexity in at least two, and at most five sentences.

```
private int foo(int x, int n) {
  int sum = 0;
  for (int i = 1; i <= n; ++i) {
    sum = sum + bar1(i, n);
  }
  return x + sum;
}
```

A1.2: Graphs (6 points)

For the following 2 questions use the following graph:
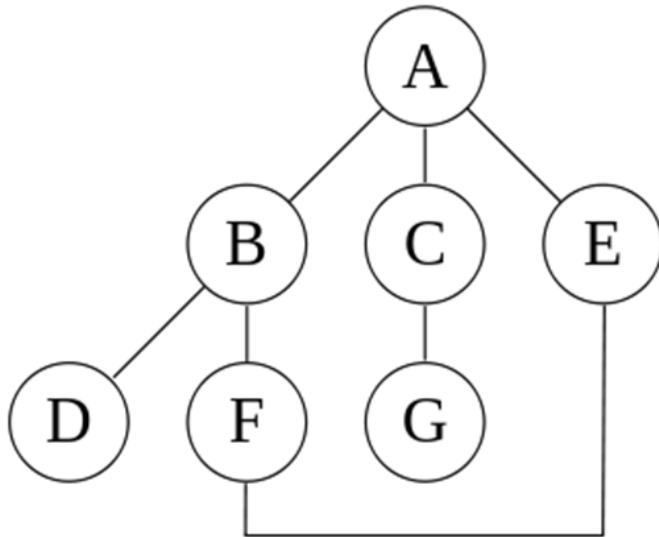
3p   **1c**   Given the above graph, write down the order that nodes are visited in, by a **breath-first search starting at node E**
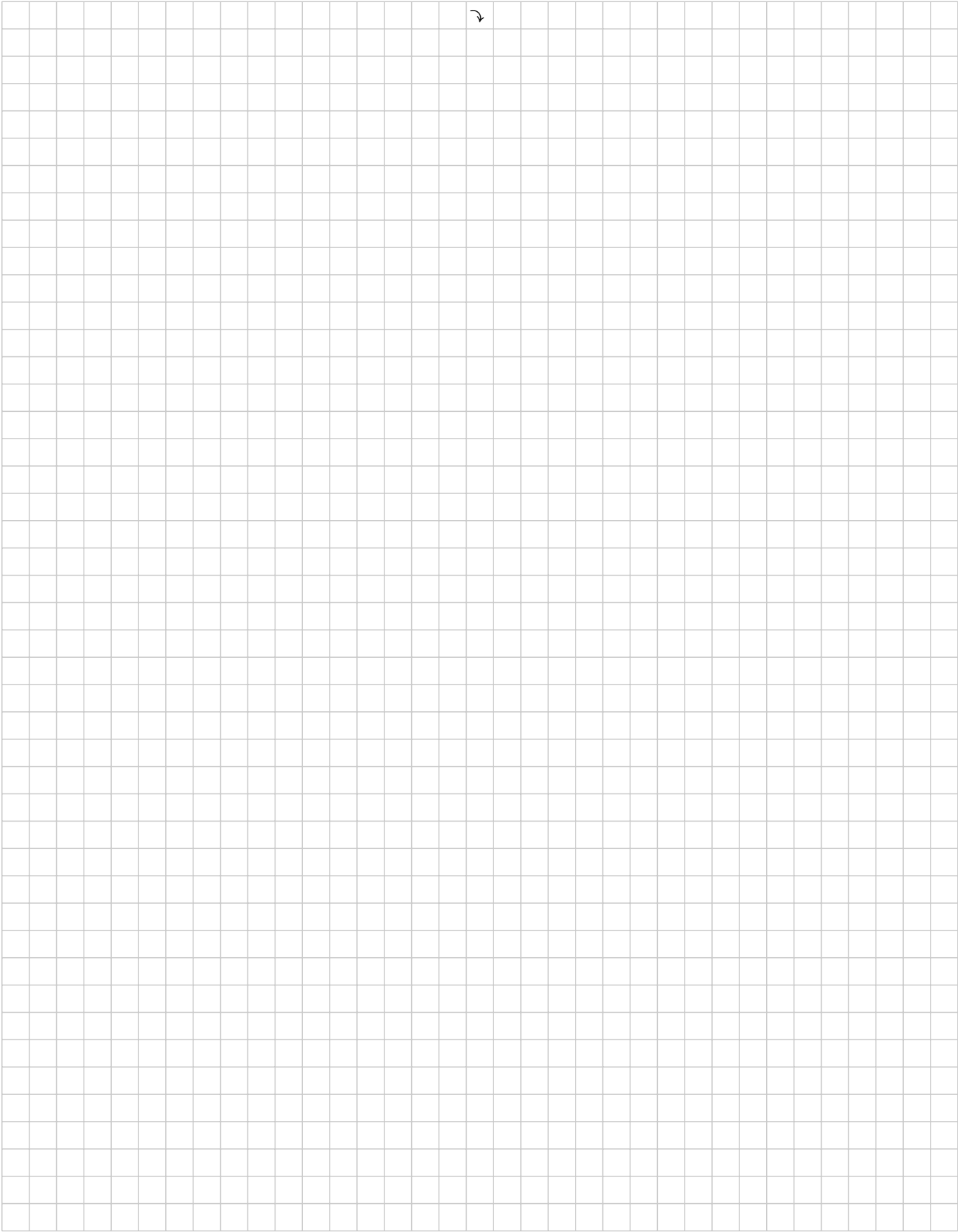
3p   **1d**   Given the above graph, write down the order that nodes are visited in, by a **depth-first search starting at node F**

A1.3: Strings (4 points)

3p **1e** Given the following words, show the process of creating a Trie. Starting with an empty Trie at the first word, draw the Trie that results after each individual word is inserted.

Exit, Abyss, Imperfect, Amoral, Extra, Amour, Imp

1p **1f** Can you save space in terms of number of nodes, by compressing the final Trie of question a? If so, draw the compressed Trie, or explain why compressing the Trie will not result in a lower space requirement in terms of total number of nodes.

A2

This section has 8 questions with a maximum of 20 points in total.

A2.1: Complexity / run time (10 points)

For the **following 5 methods**, Give the tightest possible upper bound for the worst case running time for each of the following functions in big Oh notation in terms of the variable input size **n**. For each function, your answer consists of:

1. The functions run time expresses in big Oh. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one question): $O(n^2)$, $O(n \log n)$, $O(n)$, $O(n^2 \log n)$, $O(n^5)$, $O(2^n)$, $O(n^3)$, $O(\log n)$, $O(1)$, $O(n^4)$, or $O(n^n)$
2. A short (max two sentence) text that explains how the function's run time depends on the input size **n**

*(You may assume that **System.out.println** is a single, atomic operation)*

2.5p **2a**

```
void silly(int n, int x, int y) {
  if (x > y) {
    for (int i = 0; i < n; ++i)
     for (int j = 0; j < 2 * i; ++j)
      System.out.println("y = " + y);
  } else {
    System.out.println("x = " + x);
  }
}
```

2.5p **2b**

```
int silly (int n, int m) {
  if (n < 1)
    return n;
  else if (n > 200)
    return silly (n – m, m);
  else
    return silly (n – 1, m);
}
```


2.5p **2c**

```
void silly (int n) {
  j = n * n;
  while (j > n) {
    for (int i = 0; i < n; ++i) {
      System.out.println ("j = " + j);
    }
    j = j – 7;
  }
}
```


2.5p **2d**

```
void silly (int n) {
  for (int i = 0; i < n * n; ++i) {
   for (int j = 0; j < n; ++j) {
    for (int k = 0; k < i; ++k)
     System.out.println ("k = " + k);
    for (int m = 0; m < n; ++m)
     System.out.println ("m = " + m);
   }
  }
}
```

A2.2: Sorting (6 points)

3p **2e** A programmer is wondering which sort algorithm to use in their program and has just asked you about the speed difference between a sort function that uses the insertion sort algorithm and one that uses the quicksort algorithm. Which of the following can you say? (If more than one is correct, select the most informative of them).

1. A quicksort function will always be faster than an insertion sort function
2. An insertion sort function will always be faster than a quicksort function
3. A quicksort function will usually be faster than an insertion sort function
4. An insertion sort function will usually be faster than a quicksort function
5. None of the above

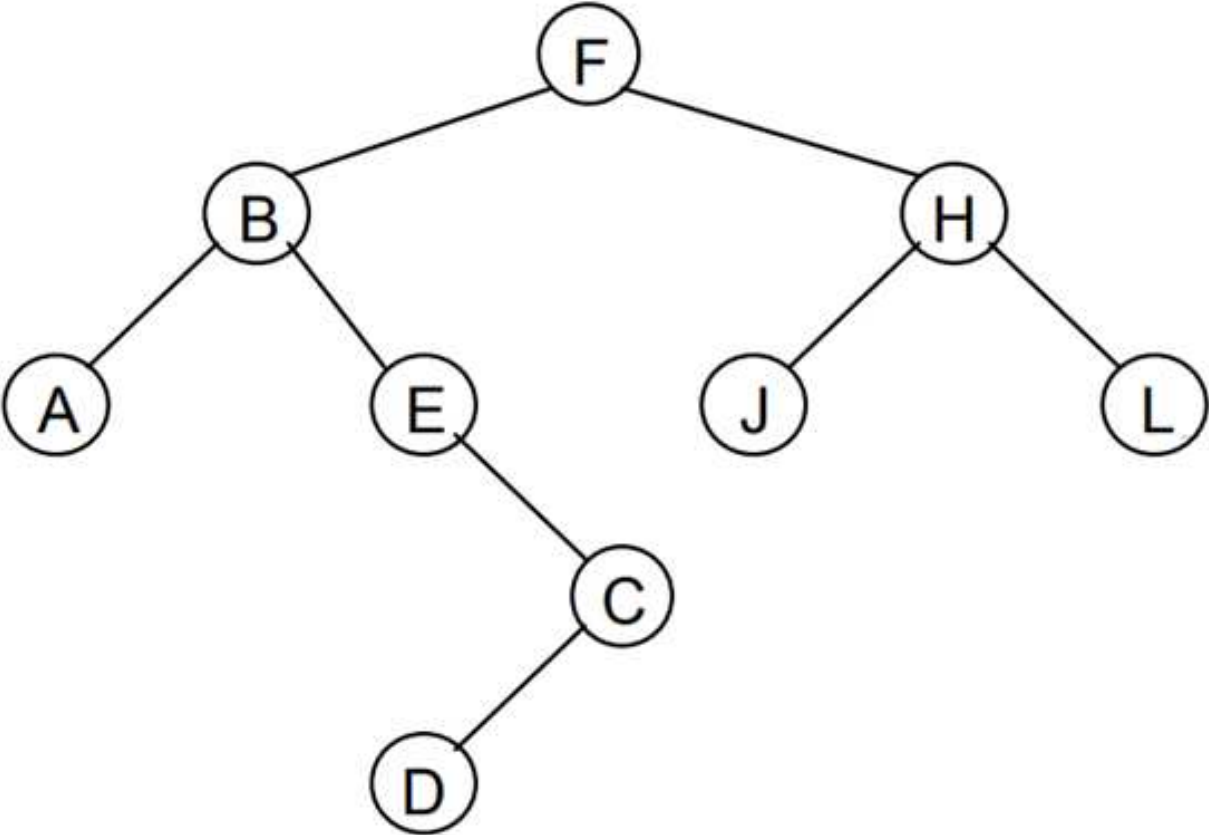Explain your answer in minimum two and maximum five sentences.

3p   **2f**   Suppose that an array contains **_n_** numbers, each of which is **1, 0,** or **-1**. Can the array can be sorted in **_O(n)_** time in the worst case? Explain your answer in minimum two and maximum five sentences.

A2.3: Trees (4 points)

2p   **2g**   What is the main advantage of an **AVL Tree** over a **Binary Search Tree (BST)**? Explain your answer in minimum two and maximum five sentences.

2p   **2h** Consider the binary tree shown below. For each of the traversals listed, give the order in which the nodes are visited.

preorder	F	B	A	E	C	D	H	J	L
inorder	A	B	E	D	C	F	J	H	L
postorder	A	D	C	E	B	J	L	H	F
breadth-first	F	B	H	A	E	J	L	C	D

Write the order of each traversal in this table

A3

This section has 7 questions with a total of 20 points.

A3.1: Complexity (10 points)

Give the tightest possible bounds for each of the following questions concerning runtime in big Oh notation in terms of the variable *n*. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one):

$O(n^2)$, $O(n \log n)$, $O(n)$, $O(n^2 \log n)$, $O(n^5)$, $O(2^n)$, $O(n^3)$, $O(\log n)$, $O(1)$, $O(n^4)$, $O(n^n)$

Assume that the most time-efficient implementation is used.

For any credit, you must explain your answer. In minimum two and maximum five questions.

2.5p **3a** Pop a value off a stack containing *n* elements implemented as an array.

2.5p **3b** Printing out all the odd values stored in a binary search tree containing *n* positive integers in ascending order.
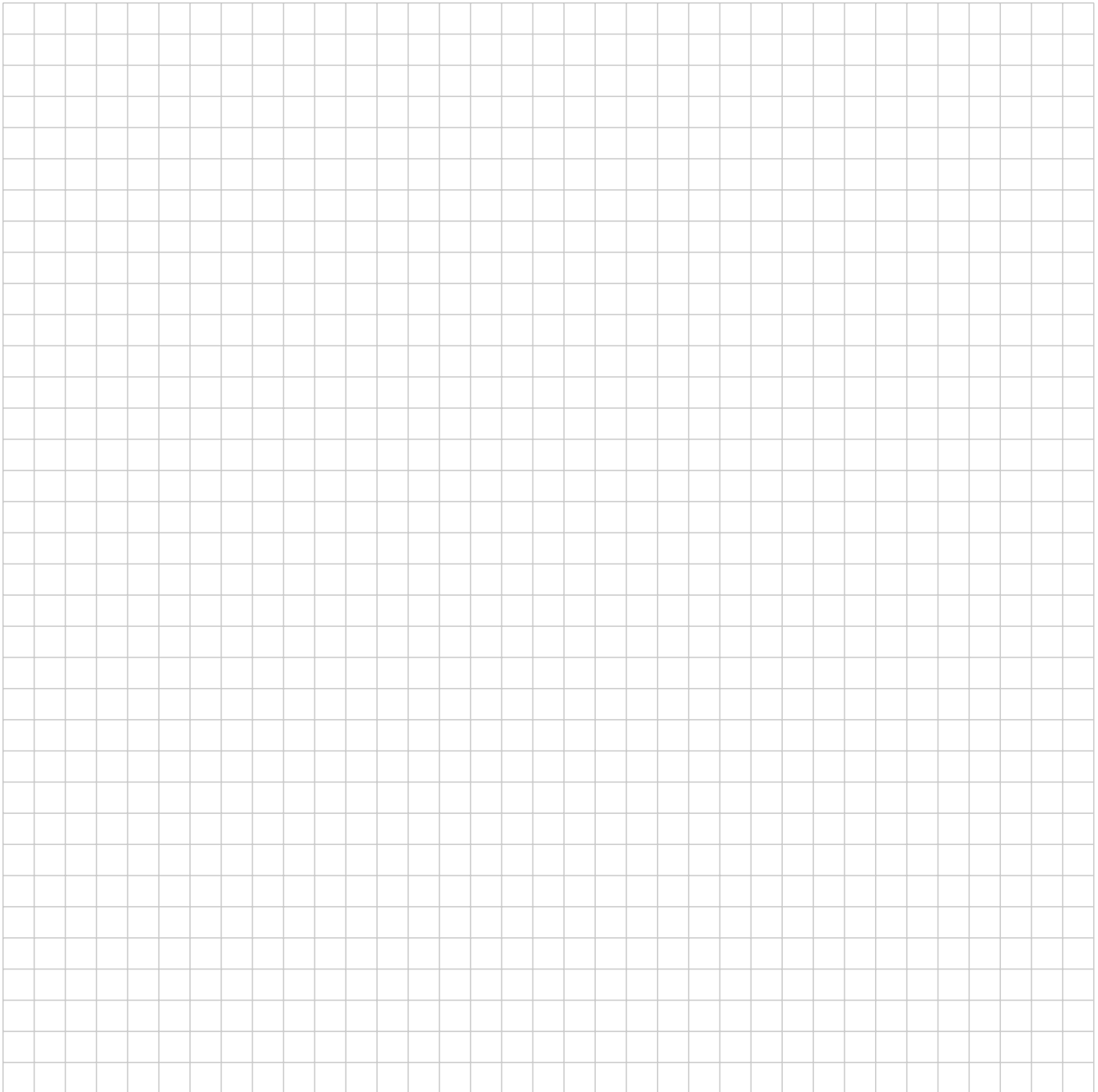
2.5p **3c** Finding the minimum value in a binary search tree with **n nodes.**

2.5p **3d** Moving **n** values from a binary min heap, into an initially empty array of the same size. The final contents of the array will be sorted from low to high.

A3.2: Hashing / Dictionaries (4 points)

4p **3e** Consider a hash table with **separate chaining** with ten hash locations. Using the hash function $h(x)$ = x **mod** 9, insert the keys *33, 54, 69, 74, 18, 19, 52, 44* (in the order given) into the hash table.

Draw the resulting hash-table in the area below.

A3.3: Heaps (6 points)

3p **3f** The method **isHeap** below determines whether a particular binary tree is a heap. The implementation this method given below contains a **few bugs**. Find these bugs and correct them.

You may rewrite the method, describe the bugs and their solutions or re-write the lines that contain the bugs. In each case, give an explanation of which bugs you found, why they are wrong and how to solve them (this also helps us to partially grade your answer).

```java
public boolean isHeap() {
 // isEmpty returns true if the heap contains no nodes
 if (isEmpty()) {
  return true;
 }

 // getKey() returns the key value of the node
 if (!leftChild().isEmpty() || leftChild().getKey() < getKey()) {
  return false;
 }

 if (!rightChild().isEmpty() || rightChild().getLey() < getKey()) {
  return false;
 }

 return leftChild().isHeap() || rightChild().isHeap();
}
```
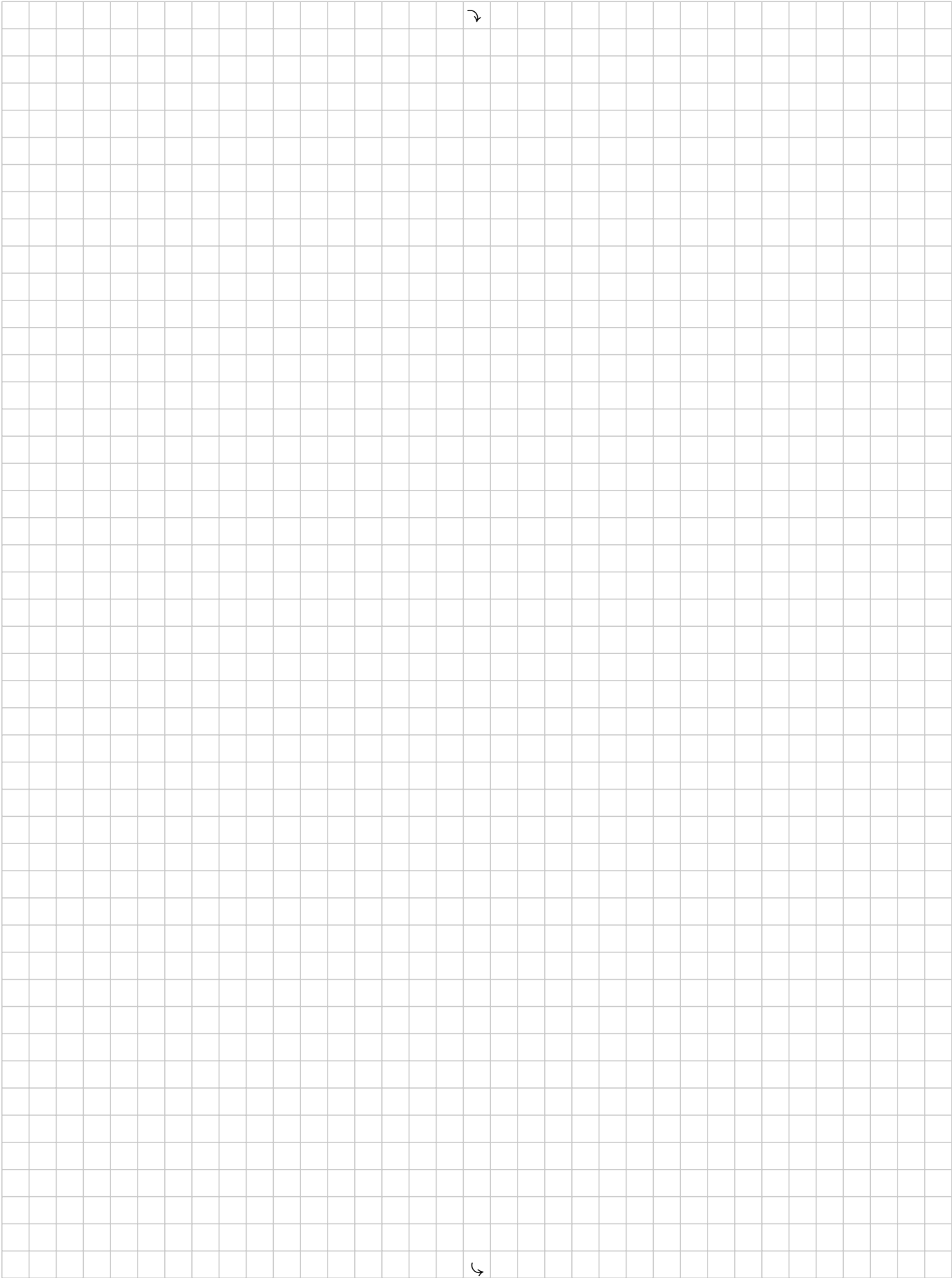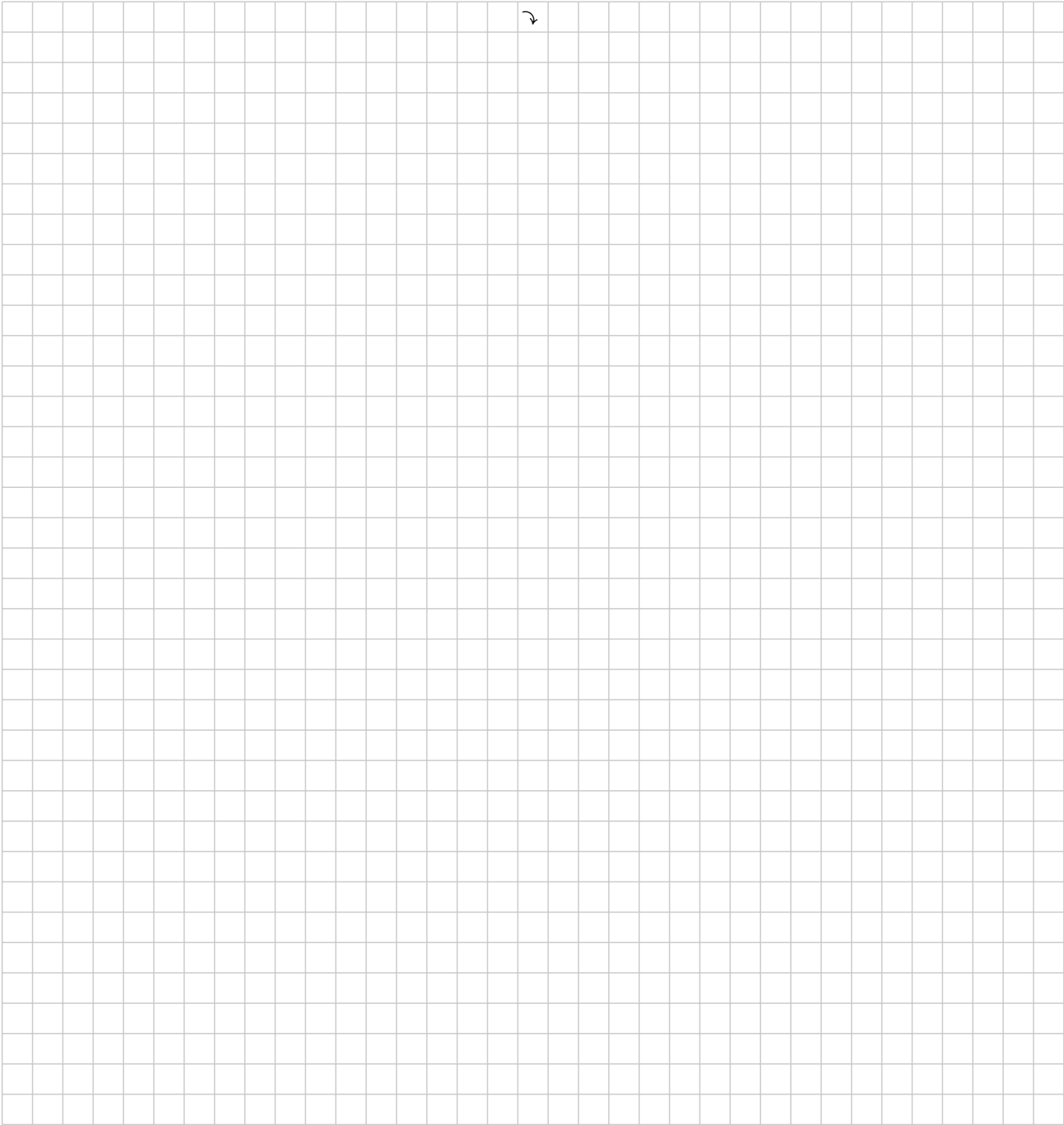
3p **3g** Draw the binary min heap that results from inserting **8, 2, 7, 6, 9, 3, 1** in that order into an initially empty binary min heap. You do not need to show the array representation of the heap.
You are only **required** to show the final tree, although drawing **intermediate** trees may result in **partial credit**. If you draw intermediate trees, please **circle your final result** for any credit.

B1

For the **B** questions give your answers in the form of **pseudo-code** or (simple) java code. You may additionally explain your answer in English text if you think it makes your solution clear or the question asks for it.

Handing in **only an English description of your algorithm will result in no points**, i.e. you **must** describe your algorithm in code to be awarded any grade.

When grading, we are not strict in terms of syntax of your code, but all steps of your solution must be interpretable and clearly explained. You may use any data-structure discussed in the course *without* describing it.

For instance:

```
LinkedList myList <- LinkedList()
myList.addAll(X)
```

is an acceptable description of an algorithm that adds all elements in X to a LinkedList.

```
BST bst <- BST()
bst.add(1)
bst.add(5)
if bst.search(4)
  print "found"
else
  print "not found"
```

Is an acceptable description of using a binary search tree, add two elements and search for an element even though it is not valid Java or Pseudocode, the idea is sufficiently clearly explained in a formal format.

Assume you defined the generic abstract data types *BinaryTree<E>* and *BinaryTreeNode<E>* as follows:

```
public interface BinaryTree {
  BinaryTreeNode getRoot();
  public void addRoot(E element);
  public boolean hasRoot();
}
```

```
public interface BinaryTreeNode<E> {
  public E getElement();
  public void setElement(E element);
  public BinaryTreeNode getParent();
  public BinaryTreeNode getLeftChild();
  public BinaryTreeNode getRightChild();
  public void addLeftChild(E element);
  public void addRightChild(E element);
  public void delete();
  public boolean hasLeftChild();
  public boolean hasRightChild();
}
```

Write a method *isBinaryTree* which takes as input a *BinaryTree<E>* instance and returns a boolean value: the returned value is true if the BST in input is a Binary Search Tree, false otherwise. To achieve the maximum grade you need to provide a solution such that, assuming that N is the number of nodes in the Tree:
- The time complexity is linear on the number of nodes, hence O(N)
- The space complexity is linear on the number of nodes, hence O(N)

You can use additional data structures defined in the course, if necessary (Arrays, List, stack, Queue, HashMap, and so on), with no need to define them.

Assume that:

- The elements in the tree are unique

- The E class implements *Comparable<E>*, hence you can use the method *compareTo* to compare two instances element1 and element 2 of the class E as in the example below:

```
if (element1.compareTo(element2)>0){
  //element 1 is bigger than element2
}else{
  //element 2 is bigger than element1
}
```

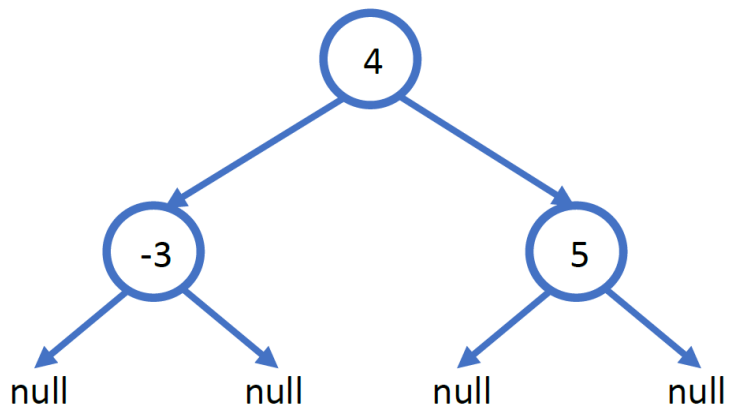Below you can find some examples:

Example 1:
Input: a BST with only one node, as in the picture below
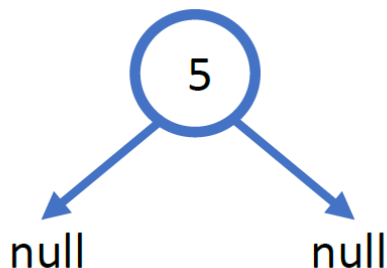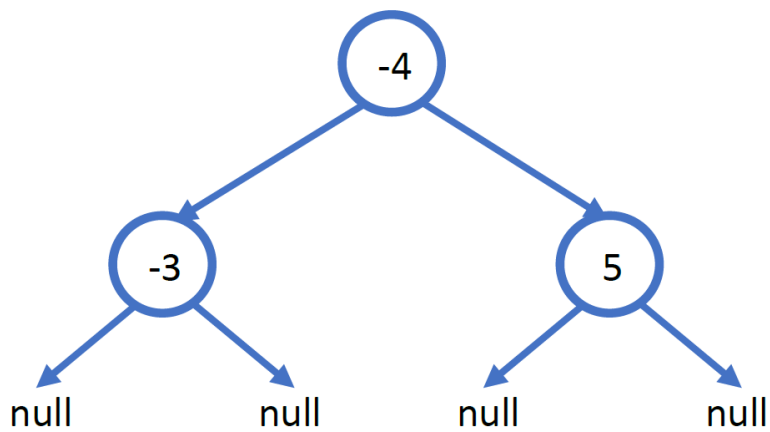

Output: true

Example 2:
Input: a BST of height 2, as in the picture below

Output: true

Example 3:
Input: a BST of height 2, as in the picture below

Output: false

20p **4**   Please provide below your implementation of the *isBinaryTree* method

B2

For the **B** questions give your answers in the form of **pseudo-code** or (simple) java code. You may additionally explain your answer in English text if you think it makes your solution clear or the question asks for it.

Handing in **only an English description of your algorithm will result in no points**, i.e. you **must** describe your algorithm in code to be awarded any grade.

When grading, we are not strict in terms of syntax of your code, but all steps of your solution must be interpretable and clearly explained. You may use any data-structure discussed in the course *without* describing it.

For instance:

```
LinkedList myList <- LinkedList()
myList.addAll(X)
```

is an acceptable description of an algorithm that adds all elements in X to a LinkedList.

```
BST bst <- BST()
bst.add(1)
bst.add(5)
if bst.search(4)
  print "found"
else
  print "not found"
```

Is an acceptable description of using a binary search tree, add two elements and search for an element even though it is not valid Java or Pseudocode, the idea is sufficiently clearly explained in a formal format.

Assume you defined the generic abstract data type *Graph<T>* as follows:

```
import java.util.List;
public interface Graph {
  public boolean adjacent(T x, T y);
  public List neighbors(T x);
  public void addVertex(T x);
  public void removeVertex(T x);
  public void addEdge(T x, T y);
  public void removeEdge(T x, T y);
}
```

Write a method *isCyclic* which takes as input a **directed** *Graph<T>* and returns a Boolean value. The returned value is true if the graph is cyclic (i.e. the graph contains at least 1 cycle), false otherwise (i.e. the graph contains no cycles).

To achieve the maximum grade you need to provide a solution such that, assuming that V is the number of verteces and E is the number of edges in the Graph:
- The time complexity is linear on the number of verteces plus the number of edges, hence O(V+E) **assuming the graph is implemented with adjacency lists**
- The space complexity is linear on the number of verteces, hence O(V)

You can use additional data structures defined in the course, if necessary (Arrays, List, stack, Queue, HashMap, and so on), with no need to define them.

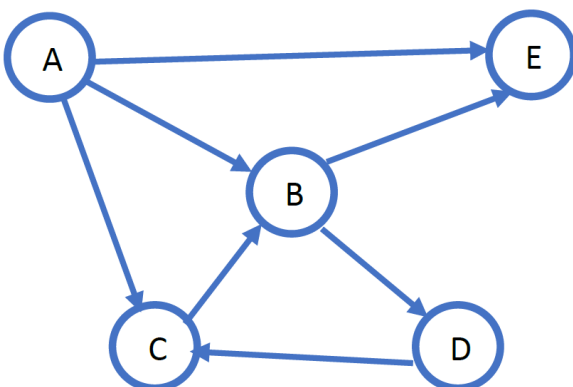Below you can find some examples:

Example 1:
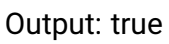Input: the graph in the picture
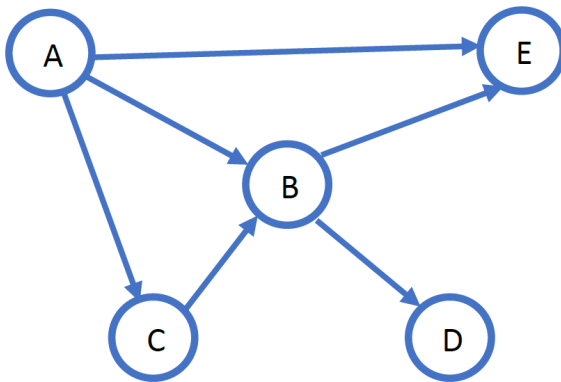
Output: true

Example 2:
Input: the graph in the picture

Output: false

Example 3:
Input: the graph in the picture

Output: true

20p **5** Please provide below your implementation of the *isCyclic* method

Extra Space

6 Use this answer box if you run out of space for any question. Make sure to make a reference to this sheet in the question's answer box.

This page is left blank intentionally