Question 1 (basic): Complexity

The following function computes the *composite* of two maps.

```
function compose(p: Map<X, Y>, q: Map<Y, Z>) → Map<X, Z>:
    r = new map with keys X and values Z
    for each key x in p:
        y = p.get(x)
        if q.containsKey(y):
            z = q.get(y)
            r.put(x, z)
    return r
```

All maps here are implemented using AVL trees. Assume that comparing two keys takes $O(1)$ time.

Let n be a number such that p and q have at most n elements.
What is the asymptotic (time) complexity of compose(p, q) in n?

Write your answer in O-notation. Be as exact and simple as possible.

Explain why the complexity of the function has this order of growth.
You can also add notes directly in the code above.

Answer

Complexity: ______________________________

Justification:

Question 2 (basic): Sorting

We run quicksort to sort the following sequence of letters alphabetically:

E	C	B	F	G	A	D

For partitioning a sequence S of values, we use the following simplified algorithm:

- The pivot is the **first element** (and does not belong to any part of the partition).
- For each part, we take the relevant elements from S in the **same order** as they occur in S.

For each call of quicksort, show:

- the sequence of values it is given (only the subarray it operates on),
- how the sequence looks after partitioning,
- what recursive calls are made,
- the sequence it returns.

Only show this when the sequence has **at least two elements**.
That is, ignore calls of quicksort on zero or one values.

Answer

Choose how you want to organise the requested information. For example, it could be a call graph (with arrows for recursive calls). You can also just use the answer table on the next page.

Question 2: Optional answer sheet

For the recursive calls, you can underline each subsequence where a recursive call is made.

Input sequence	After partitioning	Recursive calls	Output sequence
E C B F G A D			A B C D E F G

Question 3 (basic): Lists, stacks, queues

The following function modifies a list of even size.

```
function twister(l: list):
    s = new stack (using a linked list)
    q = new queue (using a circular dynamic array)

    while l not empty:
        s.push(l.removeFirst())

    while s not empty:
        q.enqueue(s.pop())

    while q not empty:
        x = q.dequeue()
        y = q.dequeue()
        s.push(y)
        s.push(x)

    while s not empty:
        l.addLast(s.pop())
```

We call the function on the following list:

A	B	C	D	E	F	G	H

Answer

How does the list look afterwards?

Question 4 (basic): Hash tables

The below open-addressing hash table uses linear probing and modular compression (mod 10).

0	1	2	3	4	5	6	7	8	9
A	B	C			D	E			F

Unfortunately, I forgot the hash codes of all the elements.
I only remember they were non-negative and smaller than 20.

For each element, determine *all* the possible hash codes.

Answer

A) ______________________

B) ______________________

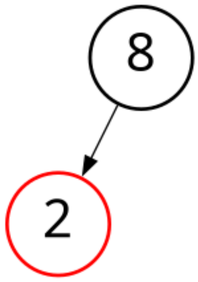
C) ______________________

D) ______________________

E) ______________________

F) ______________________

Question 5 (basic): Search trees



The drawing on the right *potentially* depicts a red-black tree (null nodes omitted).

First, decide if it is indeed a red-black tree.

- If your answer is **negative**, explain what is wrong.
  State the relevant part of the invariant and where in the tree it is broken.
- If your answer is **positive**, insert 5 into it.
  State the rebalancing steps and show the resulting red-black tree with red nodes marked.

  If you prefer, you can use the insertion procedure for 2-3 trees. In that case,
  also show the conversion to a 2-3 tree and the conversion back to a red-black tree.

Note: if you use a version of red-black trees different from your course, you must define it.

Answer

Is this a red-black tree? ☐ Yes ☐ No

Second task:

Question 6 (basic): Priority queues

You are given the following binary heap.

0	1	2	3	4	5	6	7	8
19	41	23	68	99	41	30	75	72

Answer

Is this a min-heap or a max-heap? ______________________________

Draw the heap in tree representation:

Add 36 to the heap. What is the tree representation now?

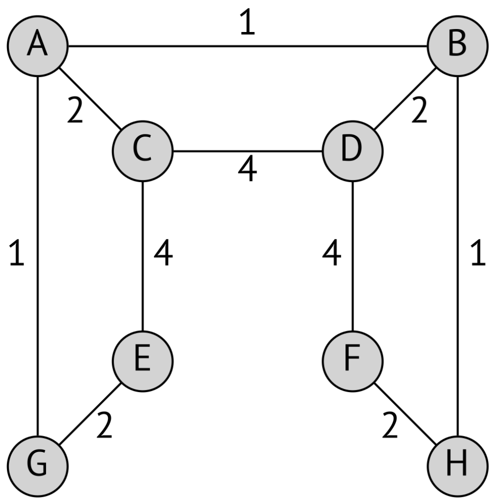
Question 7 (basic): Graphs

Perform a uniform-cost search (Dijkstra's algorithm) in the weighted graph to the right, starting from **G**.

Trace your steps in the table below. Write priority queue entries as X:c where X is the graph node and c is its cost.

Time-saving instructions:

- Do not add an entry to the priority queue if its node is already visited.
- When comparing two entries with the same cost, compare their nodes **alphabetically**.
- We are only interested in calculating distances here, not actual full shortest paths.



Answer

PQ entry to process	**Visit? yes/no**	**Priority queue (after processing entry)**
–		G:0
G:0	yes	

Question 8 (basic): Abstract data types

While surfing the web for interesting new data structures to learn, you suddenly stumble upon the following obscure code:

```
class Thermometer
    hot: Thermometer
    cool: Thermometer
    fridge: integer
    oven: string

function temperature(t: Thermometer, x: integer) → string:
    if t is null:
        return null
    if x < t.fridge:
        return temperature(t.hot, x)
    if x > t.fridge:
        return temperature(t.cool, x)
    return t.oven
```

At first, this looks very intriguing, but then you realise that you already know this data structure!

Answer

Which data structure is this? ______________________________

Find better names (in the context of that data structure) for the following:

class Thermometer = ______________________________

function temperature = ______________________________

Question 9 (advanced): Complexity

There are N cities with populations. You are given the following sorted arrays of length N:

- The array **cities** stores only the city names, but is sorted by *population*.
- The array **populations** stores the pairs of a city name and its population, and is sorted alphabetically by *name* (the first component of the pair).

Here is an example:

cities

0	Tallinn
1	Bratislava
2	Lisbon
3	Vilnius
4	Dublin
	. . .
N−2	Madrid
N−1	Berlin

populations

0	(Amsterdam, 921402)
1	(Athens, 637798)
2	(Berlin, 3677472)
3	(Bratislava, 475044)
4	(Budapest, 1706851)
	. . .
N−2	(Stockholm, 978770)
N−1	(Tallinn, 437811)

Consider the following problem:

> "Given a number K, what is the name of the largest city with population less than K?"

We want to solve this problem using a fast algorithm. What is the best asymptotic complexity in N you can achieve? Write your answer in O-notation. Justify by briefly sketching your algorithm.

- Treat number and string comparisons as taking constant time.
- You do not have to justify why there is no better algorithm.

Answer

Complexity: ______________________________

Justification:

Question 10 (advanced): Hash functions

Let T be a type (e.g., a class) with N different possible values (e.g., objects). We call a hash function for T *perfect* if it has **no collisions** and its values are the **first N natural numbers** (0, …, N-1).

Perfect hash functions are useful because they allow a very simple hash table design with good worst-case behaviour.

Here is my specification of a student:

```
class Student:
    grade: one of U, 3, 4, 5
    height: integer from 110 to 210 (that is, 110, 111, …, 209)
    nice: boolean (true or false) indicating if the student is nice
```

Design a perfect hash function for this type. It must run in constant (O(1)) time.

Notes:

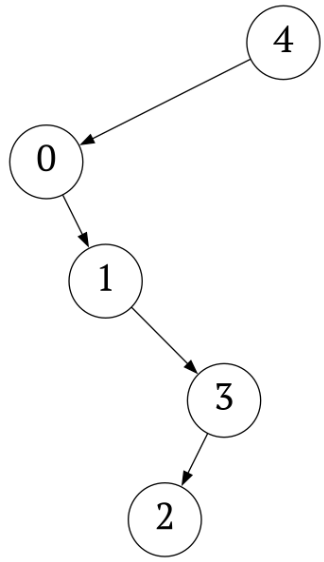
- Students with the same grade, height, and niceness are considered equal.
- You may answer using pseudocode or your favourite programming language.

Answer

Number of different possible students (N): ____________________________

function perfect_hash(s: Student) → integer:

Question 11 (advanced): Linear search trees



A tree is called *linear* if every node has at most one child that is not null. Equivalently, its height is equal to its size.

We insert the natural numbers below N (that is, 0, …, N–1) into an empty binary search tree (BST). Depending on the order of insertions, we may end up with a linear tree.

For example, for N = 5, inserting in the order 4, 0, 1, 3, 2 creates the linear BST depicted on the right (null nodes omitted).

As you know, the sequence of insertions takes $\Theta(N^2)$ time in that case. We wish to detect that case (final BST is linear) more efficiently.

Write an algorithm that takes:

- a natural number N,
- an array x of length N containing the natural numbers below N in some order

and returns a boolean. The result should be *true* exactly if inserting the values in x in that order into an empty BST would create a linear tree.

Your algorithm must have (time) complexity $O(N)$.

You may answer using pseudocode or your favourite programming language.

Answer

function test_linear(N: integer, x: array of integers) → boolean:

Question 12 (advanced): The key master

You have been taken prisoner by the key master! To escape, you need to collect all N keys, conveniently numbered in sequence (0, 1, …, N–1).

The key master has sealed each key behind a gate. That gate **may** need another key to open. After opening it, you can collect the key behind it.

Clarifications:

- The keys you collect stay with you forever. They do not get used up by opening gates.
- The same key may be needed to open several gates.

Design an algorithm to help you escape. The input consists of:

- the number N of keys,
- an array x of length N where $x[k]$ is the key needed to collect key k or *null* if no key is needed.

Print the keys in an order in which you can collect them, or raise an error if this is impossible.

But beware! If your algorithm does not run in $O(N \log(N))$ time, you will starve to death.

Notes:

- You may use any data structure or algorithm from the course in your solution.
  You don't need to implement those yourself.
- You may answer using pseudocode or your favourite programming language.
  Pseudocode can also include clear and precise natural language description.

Answer

function escape(N: integer, x: array of "integer or null"):