Question 1: Complexity

The following program takes a list of integers `x` of length k and checks if there are four elements (duplicates allowed) that sum to zero.

```
result = false
sums = new set
for a in x:
    for b in x:
        sum = a + b
        sums.add(sum)
        if sums contains -sum:
           result = true
```

What is the asymptotic complexity in k of this program if the set `sums` is implemented:

(A) using an (unsorted) dynamic array,

(B) using an AVL tree?

Answer using O-notation. In each case, your answer should be best-possible (sharp) and as simple as possible; justify that the complexity of the program is of the stated order of growth.

(replace by answer)

Question 2: Sorting

Part A: quicksort

This question is about the *partitioning* algorithm in quicksort. Consider the following array:

- [12, 3, 9, 19, 4, 2, 14, 1, 15]

Partition this array using the first element as pivot. Your answer must state or show:

- the sequence of comparisons and swaps performed,
- the resulting array, with the two partitions indicated (for example, underlined).

Note: If you use a different partitioning algorithm than the course, state the algorithm you used (either its name or a description).

Hint: You can indicate a comparison by "X < Y?" and a swap by "X ↔ Y".

(replace by answer)

Part B: merge sort

This question is about the *merge* algorithm in merge sort. Consider these two sorted lists:

- [1, 4, 9, 12]
- [2, 3, 7, 15, 16]

State the sequence of comparisons the merge algorithm performs when called on this input.

(replace by answer)

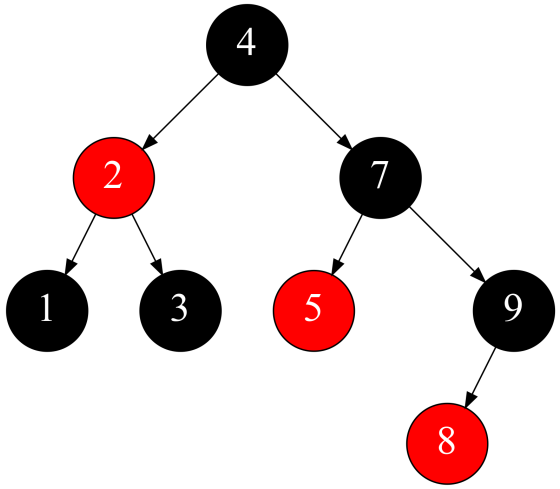
Question 3: Binary search trees

This question is about red-black trees as defined in the course. If you use a different definition of red-black trees, you must state a reference for it.

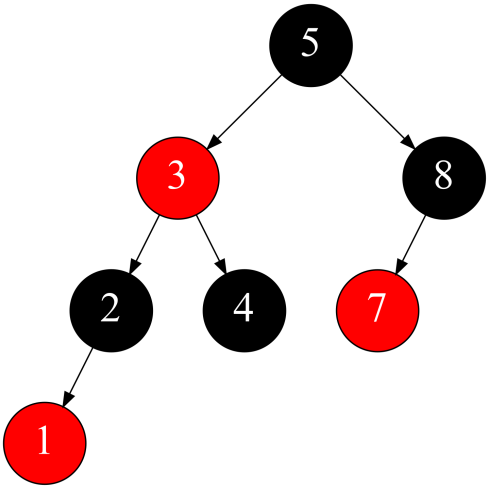
Part A

Exactly one of the following colored trees is a red-black tree. Find it. For the other ones, say why they are not red-black trees by stating what is wrong.

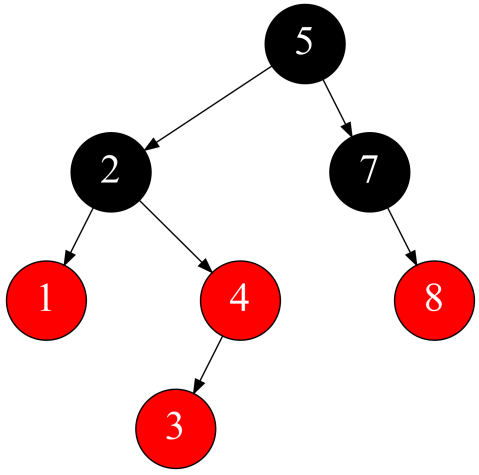
1.



2.



3.



(replace by answer)

Part B

We now insert 6 into the red-black tree from Part A. We do it in two steps. First insert it using the standard BST insertion algorithm and color it red. Draw the resulting tree:

(replace by answer)

Now rebalance to obtain a valid red-black tree. State how you do it and draw the final red-black tree.

(replace by answer)

Note: If you cannot color, draw red nodes as squares and black nodes as circles.

Question 4: Priority queues

This question is about **binary min-heaps**, just called heaps in the following, and their representation using arrays.

Part A

Exactly one of the following arrays represents a heap:

a. [17, 15, 14, 12, 9, 8, 7, 6, 5, 0]
b. [0, 5, 2, 13, 8, 14, 7, 14, 13, 16]
c. [1, 2, 4, 14, 9, 10, 13, 7, 16, 17]

Find it. For the other ones, say why they are not heaps by referencing their elements.

(replace by answer)

Part B

Remove the minimum from the heap you have identified in Part A. Your answer must state:

- the sequence of swaps performed,
- the resulting array representation of the heap.

(replace by answer)

Question 5: Hash tables

This question is about open addressing hash tables with **linear probing** and **modular hashing**. Every integer is its own hash code.

Part A

Consider the following hash table of size 9:

0	1	2	3	4	5	6	7	8
17		2		49	14	13		53

Identify the clusters. For each cluster, state all possible orders its elements could have been inserted in. Assume no deletions happened.

(replace by answer)

Part B

Insert 15 and 58 into the hash table, in this order. In each case, state the sequence of cells that are tried for insertion.

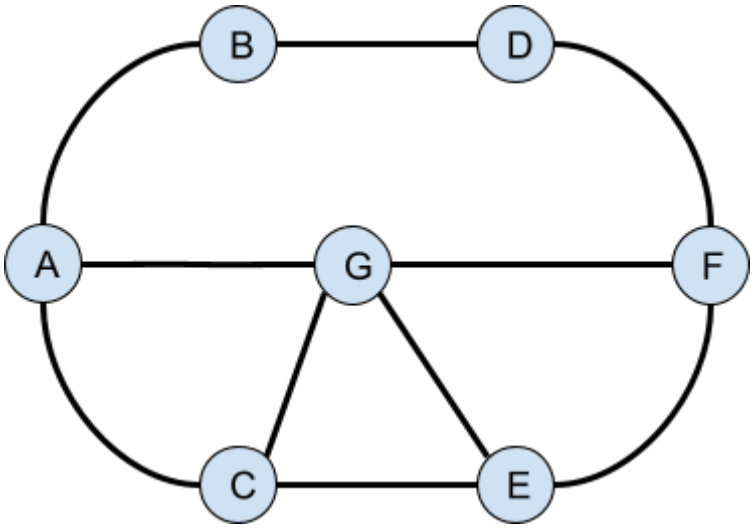
Show the final state of the hash table.

(replace by answer)

Question 6: Graphs

Part A

Consider the following graph:



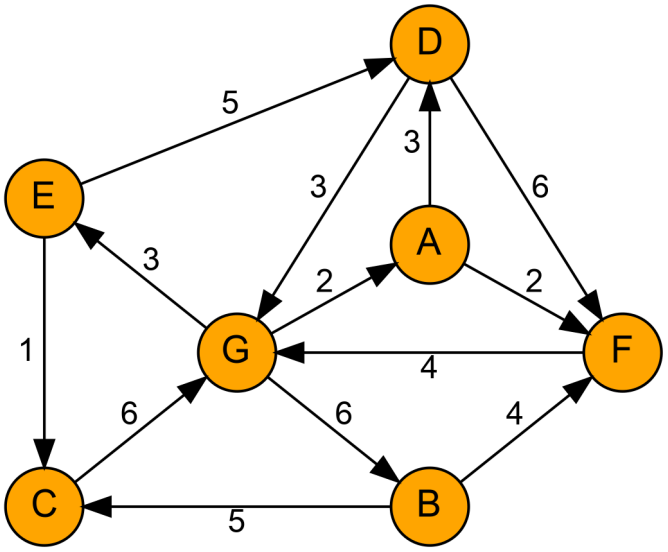
Answer the following questions:

- What is the sum of the degrees?
- Draw (or list the edges of) a spanning tree.

(replace by answer)

Part B

Consider the following weighted directed graph:



Run UCS (Dijkstra's algorithm) with starting node F. Fill in the below table with:

- the order in which the nodes are visited,
- the cost to reach them,
- the content of the priority queue (nodes with cost) after the node has been processed, omitting all entries that lead back to already visited nodes.

Note: If you use a different version of UCS than the course, give a reference.

Node	Cost	Priority queue

Question 7 (advanced): Weighted voting algorithm

In this question, you will design an efficient algorithm for ranking candidates in an election where each voter can split their vote over multiple candidates.

- A *candidate* is represented as a string. All string operations take constant time.
- A *vote* is a weighted selection of at most 5 candidates. It is represented as a map from the selected candidates to *weights*, numbers between 0 and 1. Because every vote should carry equal weight, the weights in each vote must sum to 1.
- Given a list of votes, the *score* of a candidate is the sum over all votes that select this candidate of the weight assigned to it.

Your task is to design a function

```
List<String> rankCandidates(List<Map<String, Number>> votes)
```

that takes a list of votes and returns the list of (unique) candidates appearing in the votes, sorted in descending order by their score.

Example: Given votes

- {“Mark”: 0.4, “Dora”: 0.6},
- {“Alex”: 1},
- {“Dora”: 0.5, “Mark”: 0.5},

your algorithm should return [“Dora”, “Alex”, “Mark”] because Dora has score 0.6 + 0.5 = 1.1, Alex has score 1, and Mark has score 0.4 + 0.5 = 0.9.

Your algorithm must have complexity O(n log(n)) where n is the size of `votes`. You should justify this complexity.

Note: You can specify your algorithm in pseudocode. You can use any data structure or algorithm treated in the course as a building block. You can use any reasonable names for operations on abstract data types (for example, the course API).

(replace by answer)

Question 8 (advanced): Denial-of-service attack

In this question, you will take down an online stock exchange. For this, you will send them a series of seemingly harmless requests that take increasingly longer to process. Eventually, there will be no resources left to timely serve other requests.

You have bribed an insider to supply you information about the exchange source code.

Part A

To keep unmatched orders for each stock sorted by price, the exchange stores them in an ordinary binary search tree. The specification of a buy order is as follows:

```
class BuyOrder:
    int id        // order ID (random)
    double price  // buy price in kr (high-precision)
    int number    // number of stocks to buy (at least one)

    int compareTo(BuyOrder other):
        If price != other.price:
            return Double.compare(price, other.price)
        return Integer.compare(id, other.id)
```

You can request to place buy orders via their API:

```
void placeBuyOrder(String stock, int price, int number)
```

A stock to place buy orders is “Pyramid AB”. It currently goes for 21.1337 kr, so any buy order less than 10 kr is safe: it will not match a sell order and just be stored in the BST.

Given some large number N, Describe a sequence of N requests you make in your attack. Explain in terms of asymptotic complexity the time the exchange takes to process all.

(replace by answer)

Part B

You convinced the exchange to hire you to prevent future attacks like this. Describe a change you make to their codebase that fixes the above problem.

Note: You cannot restrict order numbers per user. The high-frequency traders will get angry!

(replace by answer)

Question 9 (advanced):

Questions 7 and 8 are hard enough. They are worth 3 points instead of 2!