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.

We have two nested for-loops, each with k iterations. So the complexity is on the order of k^2 times the O-bound for the inner loop body. That O-bound is determined by the complexity of `add` and `contains`.

(A) `add` is amortized O(1). Since we execute it many times, we can regard it as O(1). `contains` is O(k^2) because `sums` grows to at most k^2 elements. So the total complexity is O(k^2 k^2) = O(k^4).

(B) sums grows to at most k^2 elements. So `add` and `contains` are O(log(k^2)) = O(log(k)). So the total complexity is O(k^2 log(k)).

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".

We initialize lo = 1, hi = 8. The pivot is 12.

- 3 < 12? Yes, we advance lo.
- 9 < 12? Yes, we advance lo.
- 19 < 12? No.
- 15 > 12? Yes, we advance hi.
- 1 > 12? No.
- We swap 19 and 1 and advance lo and hi.
- 1 < 12? Yes, we advance lo.
- 4 < 12? Yes, we advance lo.
- 2 < 12? Yes, we advance lo.
- 14 < 12? No.
- 19 > 12? Yes, we advance hi.
- 14 > 12? Yes, we advance hi.
- Finally, we swap the pivot 12 with 2.
-
-
-

The final partitioning is [2, 3, 9, 1, 4, 12, 14, 19, 15].

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.

- Comparing 1 and 2. Writing 1.
- Comparing 4 and 2. Writing 2.
- Comparing 4 and 3. Writing 3.
- Comparing 4 and 7. Writing 4.
- Comparing 9 and 7. Writing 7.
- Comparing 9 and 15. Writing 9.
- Comparing 12 and 15. Writing 12.
- Writing 15.
- Writing 16.

The output (not required to state) is [1, 2, 3, 4, 7, 9, 12, 15, 16].

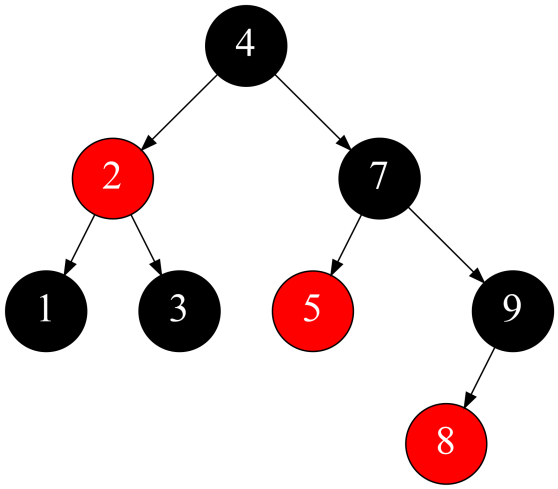
Question 3: Binary search trees

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.

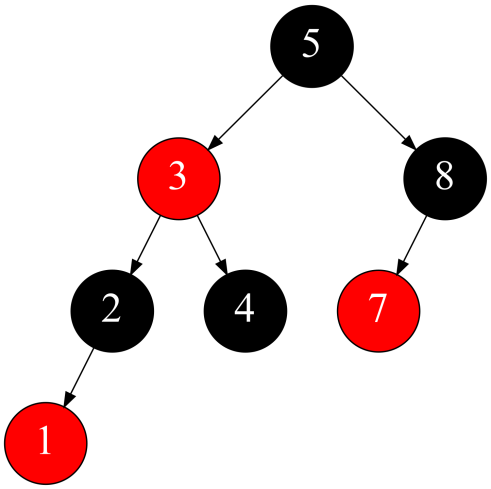
Part A

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.

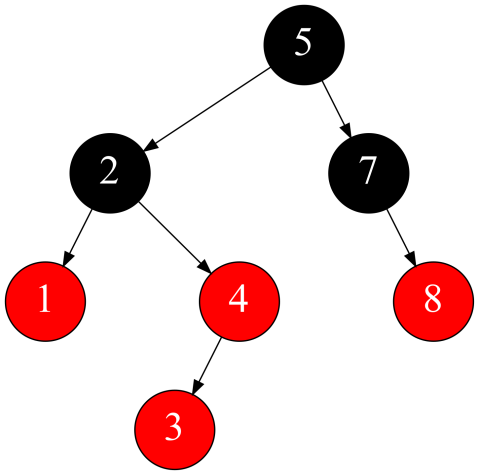
1.



2.



3.



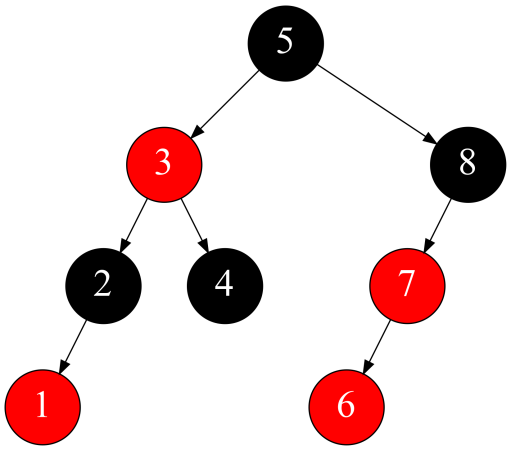
1. The black balance invariant is violated: The path 4-7-9-8 has three black nodes, but the path 4-2-1 has only two.

2. This is a red-black tree.

3. The invariant is violated: red node 3 is a child of the red node 4.

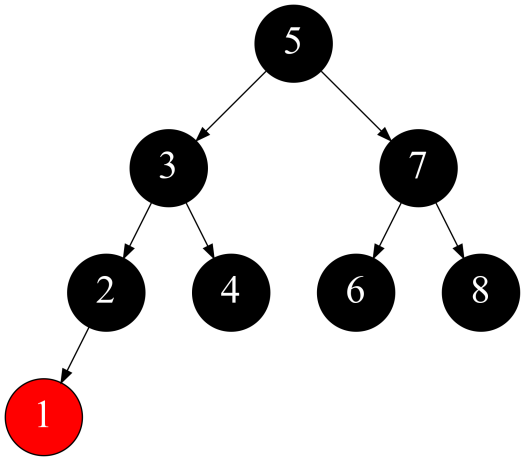
Part B

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:



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

These are the rebalancing steps (terminology from the relevant lecture):
- apply “split” to 8→7→6,
- apply “skew” to 5→7,
- apply “split” to 8→5→3,
- color root 5 black.



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.

a. This is not a heap: the root 17 is not the minimum. (Note that it is a max-heap instead.)
b. This is a heap.
c. This is not a heap: the element 7 is less than its parent 14.

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.

We swap the root 0 with the last element 16, delete the new last element 0, and then sink down the root 16:

- We swap 16 with its right child 2.
- We swap 16 with its right child 7.
-

Final state: [2, 5, 7, 13, 8, 14, 16, 14, 13]

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.

The clusters are [2], [49, 14, 13], and [53, 17]. The possible insertion orders in each cluster are:

- for [2]: only possibility "2",
- for [49, 14, 13]: "49 then 14 then 13" or "14 then 49 then 13",
- for [53, 17]: only possibility "53 then 17".

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.

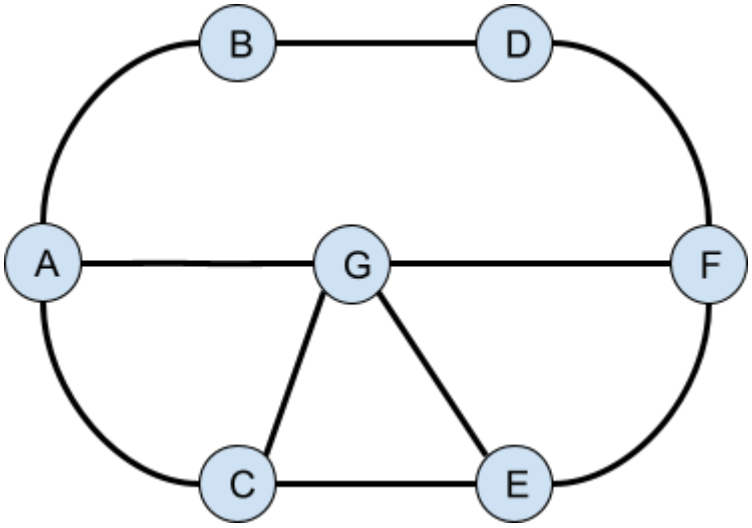
- The hash value of 15 is 6. Trying cells: 6 occupied, 7 free (inserting).
- The hash value of 58 is 4. Trying cells: 4 occupied, 5 occupied, 6 occupied, 7 occupied, 8 occupied, 0 occupied, 1 free (inserting).

0	1	2	3	4	5	6	7	8
17	58	2		49	14	13	15	53

Question 6: Graphs

Part A

Consider the following graph:



Answer the following questions:

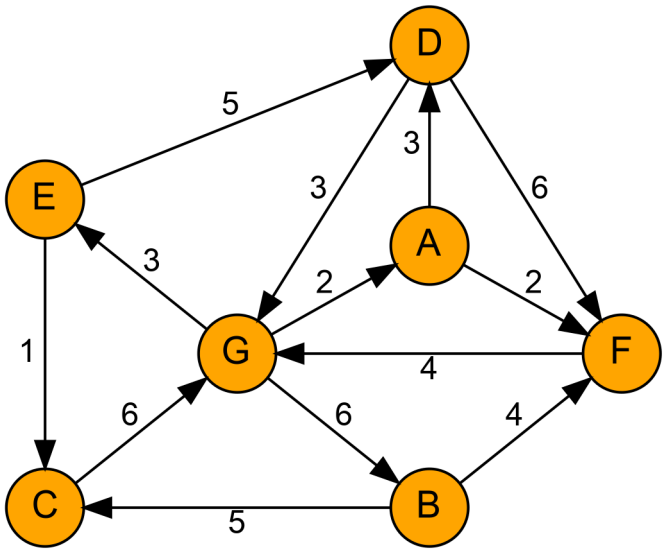
- What is the sum of the degrees?
- Draw (or list the edges of) a spanning tree.

The sum of degrees is 20 (shortcut: there are 10 edges).

Any collection of 6 edges that doesn't form a cycle will form a spanning tree. For example, we can take the path graph A-B-D-F-E-C-G.

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
F	0	G:4
G	4	A:6, E:7, B:10
A	6	E:7, D:9, B:10
E	7	C:8, D:9, B:10, D:12
C	8	D:9, B:10, D:12
D	9	B:10
B	10	

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).

There are many ways. Here is an example solution:

```
scores = new red-black map from candidates to numbers
for each vote in votes:
    for each entry (candidate, weight) in vote:
        if candidate not in scores:
            scores[candidate] = 0
        scores[candidate] += weight

x = array of entries (candidate, score) of scores
merge sort x using to reverse order of second component
return array of first components of x
```

In the nested for-loops, the first one has O(n) and the second one has O(1) (at most 5) iterations. So `scores` will have O(n) entries. The body uses red-black tree operations

O(log(n)). So this part of the program is O(n log(n)). In the second part, merge sort on an array of size O(n) is O(n log(n)). This justifies the complexity O(n log(n)).

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.

The basic idea is to cause the BST that stores buy orders to degenerate (become highly unbalanced) and behave like a linked list, realizing the worst-case complexity.

We execute `placeBuyOrder(“Pyramid AB”, i / N * 10, 1)` for i from 1 to N. This places orders with increasing prices 1/N 10 kr, 2/N 10 kr, …, N/N 10 kr (we could also place orders with decreasing prices). When the exchange stored them in the BST, this will cause the same branch to continue growing to the right. So the BST behaves like a linked list, with insertion complexity Θ(n) where n is its current size. The total complexity of processing these N orders is Θ(N^2), which for large N would mean days of processing time.

Note: There was a typo in the exam, the signature of `placeBuyOrder`enormous was given with `int price`. This was corrected during the exam via a Canvas announcement and announcements in each Zoom room. If you were trying to solve this question without this correction and got hung up on the price having to be an integer, we adjusted for that.

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!

Change the BST to a self-balanced BST (e.g., red-black tree or AVL tree).

Question 9 (advanced):

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