UNIVERSITY
OF SKÖVDE

School of Information Technology

WRITTEN EXAMINATION

Course Algorithms and Data Structures G1F

Course code IT405G Credits for written examination 6

Date 2026-01-14 Examination time 14:15-19.30

Examination responsible Yacine Atif

Aid at the exam/appendices Swedish-English / English-Swedish dictionary

Other

Instructions

- ☐ Take a new sheet of paper for each teacher.
- ☒ Take a new sheet of paper when starting a new question.
- ☒ Write only on one side of the paper.
- ☒ Write your name and personal ID No. on all pages you hand in.
- ☒ Use page numbering.
- ☒ Don't use a red pen.
- ☒ Mark answered questions with a cross on the cover sheet.

Examination results should be made public within 18 working days

Good luck!

Total number of pages: **9**

QUESTION 1 [Course Goal 1: Complexity Analysis]

a) Consider the following algorithm:

```
Input: n is a positive integer
count ← 0

// Block 1
for i ← 1 to n do
    for j ← 1 to i do
        count ← count + 1

// Block 2
k ← n
while k > 1 do
    for m ← 1 to n do
        count ← count + 1
    k ← k / 2
```

i. Analyze with explanation the time complexity of **Block 1**

ii. Analyze with explanation the time complexity of **Block 2**

iii. Compare the growth rates of Block 1 and Block 2, to determine the overall worst-case time complexity in Big-O notation, with justification

b) Consider the following algorithm:

```
Input: n is a positive integer
x ← 0

for i ← 1 to n do
    if i is a power of 2 then
        for j ← 1 to n do
            x ← x + 1
    else
        a ← i + 1
        b ← 2*i
        c ← a*b
        if c is even then x ← x + 1 else x ← x + 2
```

i. Explain how many times the condition *"i is a power of 2"* evaluates to true as a function of n.

ii. Determine the total work contributed by the else branch across all iterations i = 1 to n, considering only the operations inside the else branch. State the resulting time complexity, with explanation.

iii. Combine the results from (i) and (ii) to determine the overall worst-case time complexity of the algorithm in Big-O notation, with justification.

c) In a game tournament, each player receives a score represented as an integer. The scores of all players are stored in an array A of length n. Two players form a "pair" if the **absolute** difference between their scores is exactly k. Your task is to count the number of distinct pairs of players (i, j) with $i < j$ that form such a pair.

i. Write pseudocode for an algorithm that counts the number of such pairs by checking all possible pairs of players.

ii. Suggest a more efficient algorithm to solve the same problem without sorting the array (*Hint: think data structures*).

iii. Analyze and compare the time complexity of the algorithms in (i) and (ii). State any assumptions needed for your analysis.

QUESTION 2 [Course Goal 2: Data Structures]

a) You are given a collection of elements that must support the following operations:

- Insert an element
- Remove an element
- Check whether an element is present

i. Name **two** different data structures that can be used to store the elements. Briefly describe how each data structure supports the three operations above.

ii. For each data structure you named in (i), state the worst-case time complexity of the following operations (state any implementation-specific assumptions):

- `insertion`,
- `removal`,
- `membership_check`.

iii. Briefly explain one advantage and one disadvantage of each of the data structures you named in (i), in practice.

b) Queue, Stack, Hash and Heap:

i. Stack and Queue

Stacks and queues are container ADTs that obey different ordering principles. Show the output produced by the following program, where a **stack** and a **queue** store integers.

```
void In(int item, queue<int>& q, stack<int>& s)
{
    s.push(item);
    q.push(item);
}

void Out(queue<int>& q, stack<int>& s)
{
    s.pop();
    q.pop();
}
```

// The following utility functions display the contents of the queue and the stack:

```
void showQueue(queue<int> mq)
{
    while (!mq.empty())
    {
        cout << "\t" << mq.front();
        mq.pop();
    }
    cout << endl;
}

void showStack(stack<int> ms)
{
    while (!ms.empty())
    {
        cout << "\t" << ms.top();
        ms.pop();
    }
    cout << endl;
}
```

// Consider the following `main()` function:

```
int main()
{
    queue<int> q;
    stack<int> s;

    In(1, q, s);
    In(2, q, s);
    In(3, q, s);
    In(4, q, s);

    Out(q, s);
    Out(q, s);
```

```
    In(5, q, s);
    Out(q, s);
    In(6, q, s);

    cout << "The queue contains:";
    showQueue(q);

    cout << "The stack contains:";
    showStack(s);

    return 0;
}
```

ii. Consider a hash table of size **7** that uses the hash function:

$$h(k) = k \bmod 7$$

Show the contents of the hash table after inserting the following keys (State your collision resolution approach using a method **shown in class**):

$$\{10, 24, 31, 18, 7\}$$

iii. Construct the resulting heap after inserting all the following sequence of integers in the given order, into a priority queue that is implemented as a min-heap:

$$\{18, 4, 12, 25, 7, 20\}$$

Show all steps including the final heap tree.

c) Binary trees and binary search trees (BSTs) are fundamental data structures with different structural and performance properties. Consider inserting the following sequence of integers into an initially empty tree:

$$\{30, 20, 10, 25, 40, 50, 45\}$$

i. Insert the keys into an initially empty BST. Draw the resulting BST. (You do not need to draw intermediate steps.)

ii. Insert the same keys into an initially empty AVL tree. Draw the final AVL tree and show the rotations performed (indicate the node(s) where rotations occur and the rotation type). You do not need to redraw the entire tree after every insertion, only show the trees immediately before/after each rotation.

iii. Write the in-order traversal of the final AVL tree.

QUESTION 3 [Course Goal 3: Sorting]

a) Properties of sorting algorithms.

i. Name two different sorting algorithms studied in the class.
For each algorithm, state its worst-case time complexity.

ii. Given the array [6, 2, 8, 4, 1] show the state of the array after each complete pass of the **Bubble Sort** algorithm.

iii. Explain why Bubble Sort has a worst-case time complexity of $O(n^2)$.

b) Consider the following array of integers:

A = [9, 4, 7, 2, 6, 1]

i. Show the contents of the array after each pass of the **Insertion Sort** algorithm, until the array is fully sorted? (You do not need to show every comparison, only the array state after each pass.)

ii. Apply Quick Sort to the same array, assuming that the **first element** of the current subarray is always chosen as the pivot. Show the contents of the array **after the first partitioning step only**.

iii. Compare the worst-case time complexity of Insertion Sort and Quick Sort, where you emphasize for each algorithm the condition that leads to worst case scenarios.

c) You are asked to sort a very large array of integers that does not fit entirely in main memory. Data must be processed in large blocks read from and written to secondary storage.

i. Which sorting algorithm studied in the class is most suitable in this situation? Briefly justify your answer based on how the algorithm processes data.

ii. Explain why Quick Sort is less suitable in this scenario, even though it performs well for in-memory sorting.

iii. State the worst-case time complexity of the algorithm you selected in (i).

QUESTION 4 [Course Goal 4: Graphs].

a) Graph concepts:

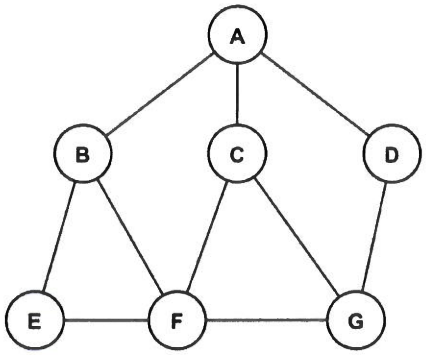
i. Briefly explain with application examples the difference between:

- a directed graph, and
- an undirected graph

ii. Name and describe two different ways to represent a graph in memory. Briefly describe one situation in which each representation is more suitable.

iii. For each graph representation you named in (ii), state the time complexity of checking whether there is an edge between two given vertices $u$and v. Briefly justify your answer.

b) Consider the following **graph** $G = (V, E)$ with vertices: $V = \{A, B, C, D, E, F, G\}$. When multiple vertices can be visited next, choose them in alphabetical order.



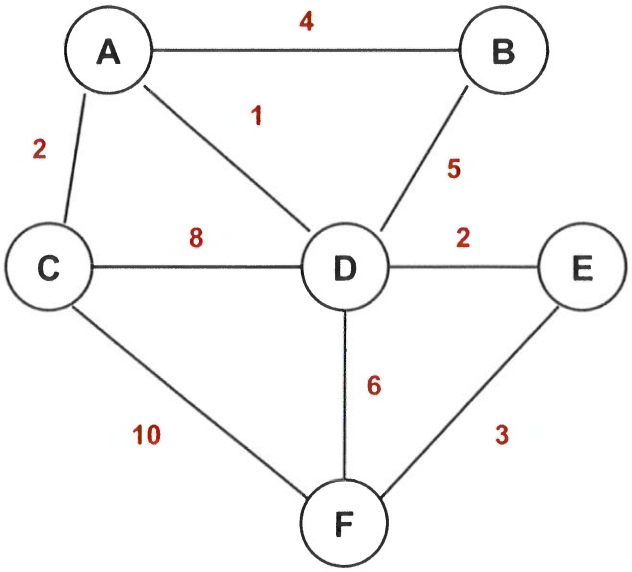
i. Starting from vertex **A**, list the order in which vertices are visited by a Breadth-First Search (BFS) traversal.

ii. Starting from vertex A, list the order in which vertices are visited by a Depth-First Search (DFS) traversal.

iii. Show one application of DFS and one application of BFS.

c) Dijkstra:

Consider the following graph.



i. Apply Dijkstra's algorithm starting from vertex A. Show the Dijkstra table with the final shortest distances and previous vertices for all nodes.

ii. Which data structure is used to implement Dijkstra's algorithm for best time complexity?

iii. Briefly explain why Dijkstra's algorithm requires all edge weights to be non-negative.

QUESTION 5 [Course Goal 5: Algorithm Design]

a) Algorithm design

i. Briefly explain the difference between:

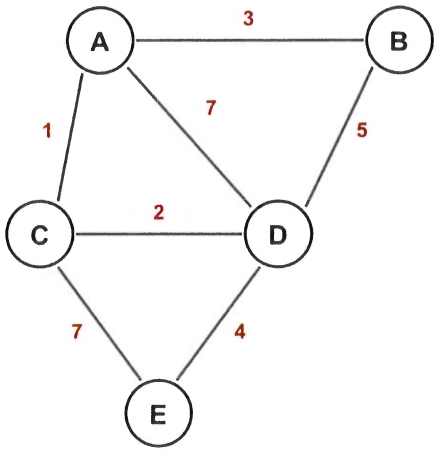
- A greedy algorithm, and
- A dynamic programming algorithm.

ii. Give **one example problem** that is typically solved using:

- A greedy approach, and
- A dynamic programming approach.

iii. Briefly explain one limitation of greedy algorithms and one limitation of dynamic programming.

b) Consider the following weighted graph with vertices: V = {A, B, C, D, E}

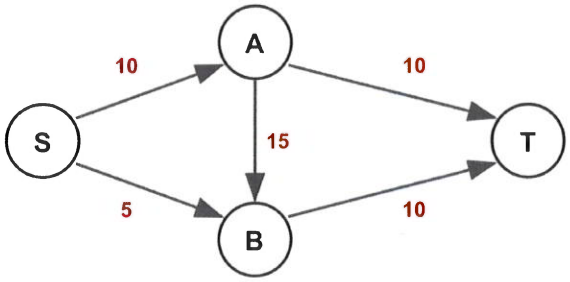


i. Apply Kruskal's algorithm to find a minimum spanning tree (MST) of the graph. Show the edges selected in the order they are added.

ii. State the total weight of the resulting minimum spanning tree.

iii. Briefly explain why Kruskal's algorithm avoids creating cycles.

c) Consider the following flow network with source S and sink T. Edge labels represent capacities:



i. Apply the Ford–Fulkerson method to compute the maximum flow from S to T. Show each augmenting path used and the flow added.

ii. State the value of the maximum flow.

iii. Briefly explain the role of the residual graph in the Ford–Fulkerson algorithm.