UNIVERSITY
OF SKÖVDE

School of Information Technology

WRITTEN EXAMINATION

Course Algorithms and Data Structures G1F

Course code IT405G Credits for written examination 6

Date 2025-01-13 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: **10**

QUESTION 1 [Course Goal 1: Complexity Analysis]

a) Derive a time complexity expression in O(·) notation for the algorithms below. The answer must contain a reasoning about the different parts of the algorithm that clearly describes how you derived the time complexity. Assuming all used variables have been declared, formulate your solution in the form of explanatory steps.

i.

```
Input: n is an integer

for j ← 1 to n do
    for k ← 1 to log(n) do
        print(j + k)
```

ii.

```
Input: n is an integer

x ← 1
for i ← 1 to n do
    for j ← i to n do
        if j mod 2 == 0 then
            x ← x * j
        else
            x ← x + 1
```

iii. *Recursive Function.*

```
function recursiveSum(arr, left, right){
    if left == right {
        return arr[left]
    }
    mid = (left + right) / 2
    return recursiveSum(arr, left, mid) + recursiveSum(arr, mid + 1, right)
}
```

b) Analyze and write the time complexity in Big O notation of the following code snippets, using the letter "n" to represent the size of the input for each of the following codes. Motivate and explain your reasoning.

i.

```
void CodeA(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j *= 2) {//Increment by powers of 2
            sum += i + j;
        }
    }
    printf("%d\n", sum);
}
```

ii.
```
void CodeB(int n, int *a) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        for (int j = n; j > 0; j--) {
            result++;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i]) {
                result++;
            }
        }
    }

    printf("%d\n", result);
}
```

iii.
```
int rec_f(int n) {
    if (n <= 0) {
        return 1;
    } else {
        return 1 + rec_f(n - 1) + rec_f(n - 2);
    }
}
```

c) Write the pseudo-ode of the function `closest_to_zero_pair(A)` that takes an array of integers A (which can contain both positive and negative integers) and returns the pair of elements whose sum is closest to zero.

i. Provide an algorithm that solves this problem in $O(n^2)$. **TIP**: Evaluate all pairs and compute the closest sum to zero.

ii. Provide another algorithm that solves this problem in O(n log n). **TIP**: Use sorting.

iii. Assume the input array may contain duplicate values. How should the above algorithms be adjusted to handle duplicates? Does it change the time complexity?

QUESTION 2 [Course Goal 2: Data Structures]

a) Consider a hashtable with an array of size n, where elements are stored using **linear probing** for collision resolution. The hash function is defined as:

$$h_1(k) = k \bmod n$$

The following operations are supported:

`insert(hash_table, key)`: Inserts the key into the hashtable.
`search(hash_table, key)`: Searches for the key in the hashtable.

i. Use **linear probing** to resolve collisions in a hashtable of size n = 7, and show the final state of the hashtable, after all of the following insertions, if successful:

- Insert 10
- Insert 17
- Insert 24
- Insert 31
- Insert 45

ii. Modify the hash function to include double hashing:

The second hash function is defined as:

$$h_2(k) = 1 + (k \bmod (n-1))$$

Consider a **new** hastable of size n = 7 that employs double hashing to resolve collision, using the function h_1 used in the previous question and the above function h_2 that are applied in sequence when collision occurs. Show the state of the table after all the following insertions, if successful:

- Insert 10
- Insert 17
- Insert 24
- Insert 38

iii. An alternative collision resolution mechanism uses **chaining**. Show the final state of the hashtable after inserting the following keys: {10,17,24,38,45}, when chaining mechanism is applied.

b) Queue, Stack and Heap:

i. Determine whether the following statement is true or false. If false, explain why. In a **linked-list implementation of a Queue ADT**, both enqueue and dequeue are O(1) time complexity operations.

ii. Compare the time complexity of the push() and pop() operations for both array and linked list implementations of a stack.

iii. Consider the following sequence of numbers to insert into a **binary heap,** in the given order:

{15, 10, 30, 40, 50, 20, 5}

Construct a **min-heap** by drawing the binary tree representation of the heap.

c) Trees

i. Draw the BST after inserting the elements {10, 50, 30, 20, 40} into a Binary Search Tree (BST).

ii. Insert the elements {50,40,30,20,10} into an AVL tree in this order. Consider the folloings when doing so:

- Draw the tree after each insertion.
- Show and explain all balancing steps (rotations) to maintain the AVL property.
- **Write the final tree's in-order traversal**.

iii. Given the following sequence of operations on an empty Splay Tree:

- Insert 25
- Insert 15
- Insert 50
- Search for 15
- Insert 35
- Search for 50

Show the Splay Tree structure after each operation and circle the final one.

QUESTION 3 [Course Goal 3: Sorting]

a) Selection Sort

i. Explain the Selection Sort algorithm and describe its primary characteristic in terms of how it selects elements for sorting.

ii. Given the array [4, 7, 1, 5, 3], show the state of the array after each iteration of the outer loop in the Selection Sort algorithm.

iii. Provide with explanation the time complexity of Selection Sort and explain why it is preferred for sorting very small datasets?

b) Mystery Sort

Consider the following code:

```
Function MysterySort(Array A, Integer n)
    For i from 1 to n-1
        key = A[i]
        j = i - 1

        While (j >= 0 and ...C...)
            A[j + 1] = A[j]
            j = j - 1
        EndWhile

        A[j + 1] = key
    EndFor
EndFunction
```

i. What sorting algorithm does the above code represent?

ii. Provide the missing condition **C** in the while loop.

iii. Analyze with explanation the best-case and worst-case time complexities of Mystery Sort.

c) Merge Sort

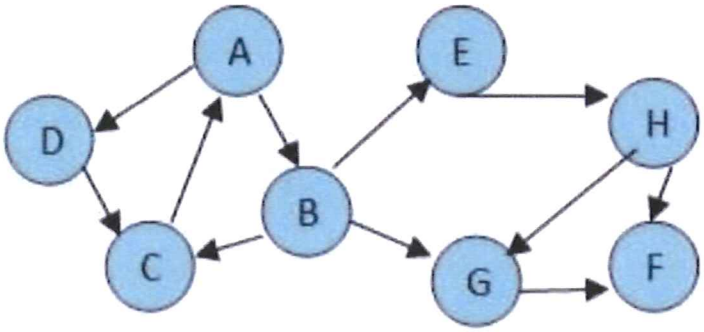
i. Explain the Merge Sort algorithm and how it uses the divide-and-conquer approach.

ii. Given the array [12, 7, 14, 9, 10, 11, 8], show the recursive splitting process. Indicate how subarrays are combined to produce the final sorted array.

iii. Why is Merge Sort a good choice for sorting linked lists compared to arrays? Explain your answer by comparing the access patterns and memory usage.

QUESTION 4 [Course Goal 4: Graphs].

Given the following undirected graph where nodes are labeled A, B, C, D, E, F, G, H , with edges shown below. Assume that when choosing between multiple neighbor nodes, we always select the **smallest letter in alphabet order**.



a) Graph Properties:

i. State whether each of the following statements is TRUE or FALSE:

- The graph is **complete**.
- The graph is **directed**.
- The graph is **connected**.
- The graph is **weighted**.

ii. Breadth-First Search (BFS)

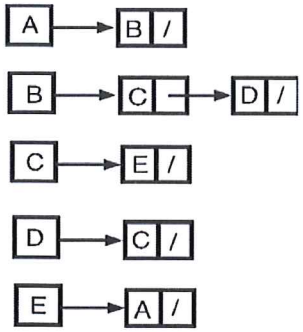
Starting from Node A, write the order of nodes visited using the BFS algorithm

iii. Depth-First Search (DFS)

Starting from Node A, write the order of nodes visited using the DFS algorithm.

b) Graph Representation

The adjacency list representation of a graph with five vertices A, B, C, D, E is given below.



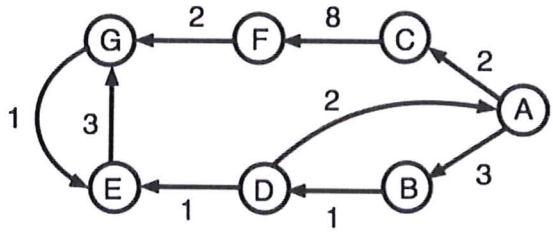
i. Provide the adjacency matrix.

ii. Draw the graph.

iii. Discuss the adjacency list and matrix representation of the graphs in terms of complexity analysis.

c) Dijkstra:

Consider the following graph.



i. Starting from Node A, show Dijkstra table that illustrates the order of visits of the above graph when running the algorithm till the end.

ii. Show the resulting shortest paths from Node A to every other node, and corresponding distances.

iii. Discuss, with justification, the Abstract Data Type (ADT) used to implement Dijkstra's algorithm. Explain why this ADT is suitable and how it influences the efficiency of the algorithm.

QUESTION 5 [Course Goal 5: Algorithm Design]

a) Algorithm Design

Consider a function *f* where n is an integer (n ≥1):

$$f(n) = f(n-1) + 2 \cdot f(n-3) \quad \text{for } n > 3$$

$$f(1) = f(2) = f(3) = 1$$

i. Write a recursive pseudo-code to compute the function ff without using dynamic programming.

ii. Using the recursive method, compute f(5) and f(6).

iii. Describe any relevant observations regarding the **time complexity** of the recursive computation and explain why it may be inefficient.

b) Dynamic programming

i. Using the **"Bottom-Up"** method, resolve the above problem (a) and compute f(5) and f(6). Explain your solution approach.

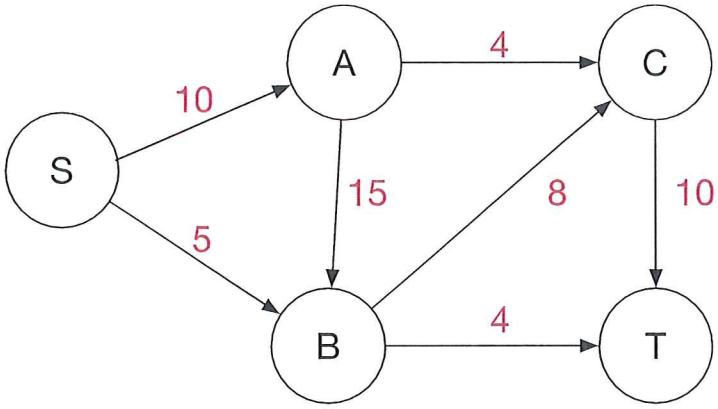
ii. Using the **"Top-Down"** method with memoization, resolve the above problem (a) and compute f(5) and f(6). Describe your approach..

iii. Compare the "Bottom-Up" and "Top-Down" methods in terms of complexity and ease of implementation.

c) MaxFlow

In the context of network flow problems, consider a **directed graph** representing a traffic network where each edge has a capacity indicating the maximum number of vehicles that can pass through per unit time.

i. Given the following directed graph with capacities on the edges, use the Ford-Fulkerson algorithm to find the maximum flow from S to T. Show the flow in each step and the residual capacities.



ii. Calculate the total flow from S to T after applying the Ford-Fulkerson algorithm.

iii. Explain the concept of an augmenting path in the context of the Ford-Fulkerson algorithm and its significance in determining the maximum flow.