UNIVERSITY
OF SKÖVDE

School of Information Technology

WRITTEN EXAMINATION

Course	Algorithms and Data Structures G1F		
Course code	IT405G	Credits for written examination	6
Date	2024-03-01	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 answer sheet of paper when starting a new question.
- ☒ Write only on one side of the answer sheet 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) Given the functions *f(n)* and *g(n)*:

i. if $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$, provide the final expression using a single Big O notation of $T_1(n) + T_2(n)$

ii. if $f(n) \leq g(n)$, provide the final expression using a single Big O notation of $O(f(n) + g(n))$

iii. if $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$, provide the final expression using a single Big O notation of $T_1(n) \times T_2(n)$

b) Use the Big O notation to indicate the complexity of the following algorithms with the letter "n" representing the size of the input. Then, sort them in ascending order from the lowest to the highest complexity order (a, b, …).

i. (A) Logarithmic, (B) Constant, (C) Linear
ii. (A) Quadratic, (B) Exponential
iii. (A) Linearithmic, (B) Cubic

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

i.

```
void Code1(double *a){
double max = a[0];
int i;
for(i=1; i<n; i++)
 if(a[i]>max)
  max = a[i];
}
```

ii.

```
void Code2(int n, int *a){
int sum = 0, val = 0, f[n], i, j;
for(i=0; i<n; i++){
  for(j=i+1; j<n; j++){
   if(a[i]< a[j])
    val= 1;
   else
    val= 0;
   sum = sum + val;
  }
  f[i]= sum; sum = 0;
}
```

iii.

```
int Code3(int *a, int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (a[mid] == target)
            return mid;
        else if (a[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

QUESTION 2 [Course Goal 2: Data Structures]

a) Consider an array-based implementation of a Stack with a capacity of 5 elements. Initially, the stack is empty.

i. What is the status of the stack, presenting the elements in the stack from bottom to top, after completing this sequence of operations:

```
push(1)
push(2)
push(3)
pop()
push(4)
peek() // get_top()
push(5)
push(6)
pop()
push(7)
```

ii. Considering **a new** empty stack, describe the final state of the stack after executing the following pseudo-code:

```
for i = 1 to 6
    if i is even
        push(i)
    else if i is odd and the stack is not empty
        pop()
push(7)
while the stack is not full
    push(peek() + 1)
```

iii. Fill-in the gaps of the following C-code illustrating a linked-list implementation of the stack:

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct Stack {
    ______(a)______ top;
} Stack;

void push(Stack* stack, int data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    if (newNode == NULL) {
        exit(1); // Failed allocation
    }
    newNode->data = data;
    newNode->next = ____(b)_______;
    stack->top = newNode;
}

int pop(Stack* stack) {
    if (stack->top == NULL) {
        return -1; // Indicate empty stack
    }
    Node* temp = stack->top;
    int popped = temp->data;
    stack->top = ______(c)_______;
    free(temp);
    return popped;
}

int peek(Stack* stack) {
    if (stack->top == NULL) {
        return -1; // Indicate empty stack
    }
    return _____(d)______;
}
```

b) A Priority Queue with the min-heap property is considered. This means that the underlying data structure of the priority queue ADT is a binary heap. You are expect to show the status of the heap after inserting each element into the queue (one "drawing" per insertion). The queue is initially assumed to be empty.

i. Draw the status of the heap after completing the following sequence of operations:
`Insert 21; Insert 31; Insert 24; Insert 65; Insert 6; Insert 1; Insert 32; Insert 16`

ii. Continuing from the previous sequence of insertions, show the status of the heap after completing the following sequence of operations:
`Insert 68; Insert 19; Insert 13`

iii. Continuing from the previous sequence, show the status of the heap after invoking `removeMin()` three successive times.

c) Trees

i. Draw the AVL tree that results from inserting the keys: 2, 3, 5, 9, 6, 4 in that order into an initially empty AVL tree. Show all insertion steps and **circle** your answer reflecting the final AVL tree.

ii. Show the resulting Splay tree after inserting these keys: 2, 3, 5, 9, 6 in sequence, and then searching Node 10. Show all insertion steps and circle your final Splay tree answer.

iii. Analyze the advantages and disadvantages of each tree type (AVL and Splay) in terms of insertion complexity, search efficiency, and balance maintenance.

QUESTION 3 [Course Goal 3: Sorting]

a) Consider as input the unsorted array: 10, 9, 11, 4, 7, 8, 5, 6, 3, 2; which is to be sorted in ascending order using the **Insertion Sort** algorithm.

i. Show the status of the array after completing each iteration of the **outer** loop of the algorithm, controlled by the index *i*. That is, when i=0, i=1, i=2, etc.

ii. How many comparisons among the array elements were required for this input?

iii. Derive with explanation the worst-case time complexity in Big-O notation of insertion sort.

b) Consider as input the unsorted array: 10, 9, 11, 4, 7, 8, 5, 6, 3, 2; which is to be sorted in ascending order using **Selection Sort** algorithm.

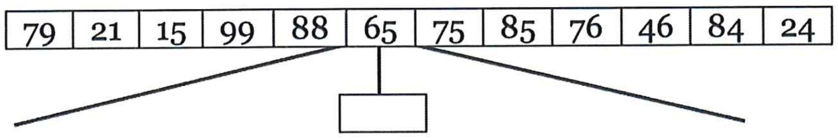
i. Show the status of the array after completing each iteration of the **outer** loop controlled by the index *i*. That is, when i=0, i=1, i=2, etc.

ii. How many comparisons among the array elements were required for this input?

iii. Derive with explanation the worst-case time complexity in Big-O notation of selection sort.

c) Consider as input the unsorted array: 79, 21, 15, 99, 88, 65, 75, 85, 76, 46, 84, 24 which is to be sorted in ascending order using **Quick Sort** algorithm:

i. Draw the corresponding tree showing all left and right partitions, as well as middle elements (pivots), from root to all leaves, when applying the division phase of the algorithm. Each partition must be in the correct order (and position), derived by the algorithm. For this example, use as pivot the element in the middle, applying "floor" when odd partitions are encountered.

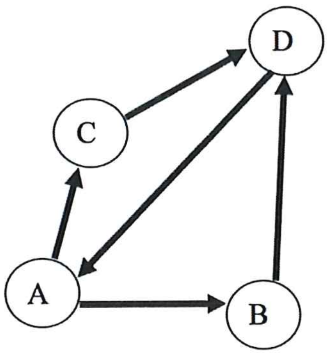


ii. Explain what the **partition** function does in the context of QuickSort.

iii. What is the worst-case time complexity of QuickSort, and under what circumstances does it occur?

QUESTION 4 [Course Goal 4: Graphs].

a) Consider the following graph.

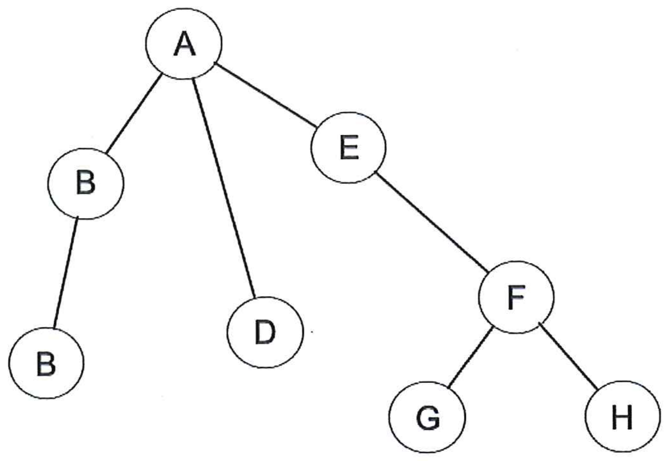


i. Draw the adjacency matrix.

ii. Draw the list representation for the above diagraph.

iii. Determine the worst case running time complexity in Big-O complexity (worst-case) to find the in-degree of a node in the list-representation of the diagraph (use V representing vertices and E representing edges, rather than n in your answer). Motivate your answers with explanations.

b) Consider the following graph:

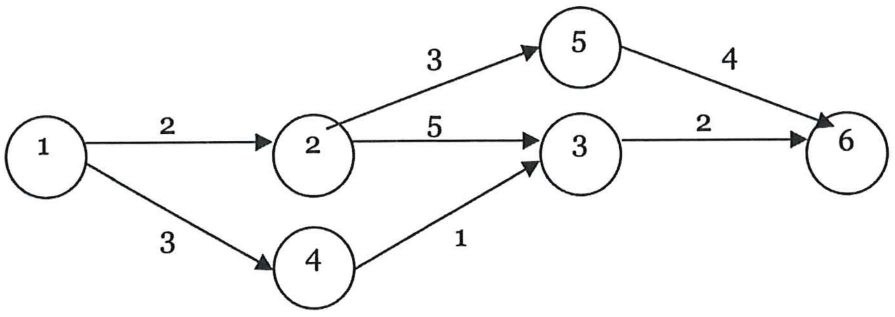


i. The Graph is (True or False): A. Complete? B. Directed? C. Weighed? D. Connected?

ii. Specify the order of depth-first search visits starting from Node A in the above graph (DFS traversal).

iii. What is the order of visited nodes using BFS algorithm (starting from Node A)?

c) Consider the following graph:



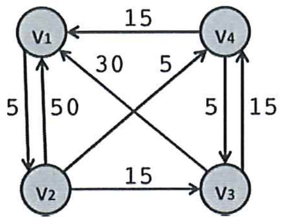
i. Use Dijkstra's algorithm to determine the shortest path from Node 1 to each of the other vertices in the **next** graph below. Show your Dijkstra table.

ii. What is the resulting path and its weight.

iii. Provide two real-world applications of Dijkstra algorithm.

QUESTION 5 [Course Goal 5: Algorithm Design]

a) Consider the following graph.



i. What Floyd-Warshall algorithm is used for?

ii. Provide the initial matrix D_0 in Floyd-Warshall algorithm considering the above graph.

iii. Construct subsequent Matrices D_1 to D_4. and interpret the results.

b) In the following Huffman Coding related question, consider the space (*sp*) as a character and motivate your answers with explanation.

i. Construct <u>the tree</u> developed by the Huffman Coding algorithm to create the prefix code for the message: **I CAN MAKE IT**.

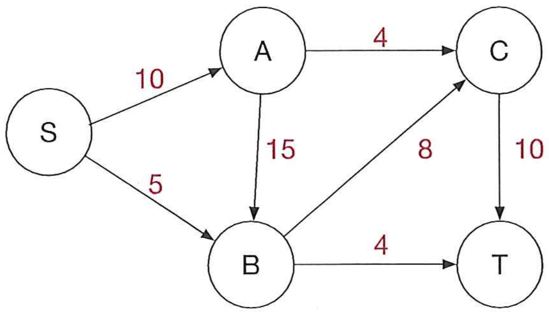
ii. Indicate the associated code for each character and the prefix code.

iii. Provide the total cost of such code.

c) MaxFlow

In the context of network flow problems, consider a directed graph representing a network of water pipes. Each edge in this graph has a capacity, which is the maximum amount of water that can flow through that pipe per unit time. Your task is to determine the maximum total amount of water that can flow from a source node (S) to a sink node (T) in the network.

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.