UNIVERSITY
OF SKÖVDE

School of Information Technology

WRITTEN EXAMINATION

Course Algorithms and Data Structures G1F

Course code IT405G Credits for written examination 6

Date 2024-01-12 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) 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 i ← 1 to 3n
    print(n)

for i ← 1 to n
    for j ← i to n
        print(i*j)
```

ii.

```
Input: n is an integer

x ← 1

for i ← 1 to n
     if i is even then
        for j ← 1 to i
            x ← x + 3
     else
        x ← x + 2

for i ← 1 to n
      for j ← 1 to n
         for k ← 1 to j
             x ← x+1
```

iii.

```
function f(array, key, left, right){

    if right < left {
        return false
    }

    len = right - left
    c = left + len / 2

    if getElementAt(array, c) > key {
        return f(array, key, left, c - 1)
    } else
        if getElementAt(array, c) < key {
        return f(array, key, c + 1, right)
        }
    return true
}
```

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 *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;
    }
  }
```

ii.
```
void CodeB(int n, int *a) {

      int i, j, k, count = 0;

      for (i = 0; i < n; i++) {
          for (j = n; j > 0; j--) {
              count++;
          }
      }

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

      printf("Count: %d\n", count);
  }
```

iii.
```
void CodeC (int n) {

        int i, j, count = 0;

        for (i = 1; i <= n; i++) {
          for (j = 1; j <= i/2; j++) {
              count++;
          }
        }

      printf("Count: %d\n", count);
   }
```

c) Write a function smallest_difference_pair(A) that takes an array of distinct integers A and returns the smallest absolute difference between any two distinct elements in the array. In case there are multiple pairs with the same smallest difference, the function should return any one of those pairs.

i. Provide an algorithm that solves this problem in $O(n^2)$.

ii. Provide another algorithm that solves this problem in O(n log n).

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

QUESTION 2 [Course Goal 2: Data Structures]

a) Consider the following function **magic_queue(int n)**, written in C-like pseudo-code which indicates the available Queue and Stack ADT operations :

```
Queue create_queue (int n) {
    Queue Q = new Queue()
    For i from 1 to n {
        Q.enqueue(i)
    }
    Return Q
}

Queue do_something(Queue Q) {
    Stack S = new Stack()

    While not Q.isEmpty() {
        S.push(Q.dequeue())
    }

    While not S.isEmpty() {
        Q.enqueue(S.pop())
    }

    Return Q
}

void print_queue(Queue Q) {
     While not Q.isEmpty(){
        Print(Q.dequeue())
     }
}
```

i. What is the output when running the following sequence of function calls:

```
Queue Q = create_queue(5);
do_something(Q);
print_queue(Q);
```

ii. What is the output when running the following sequence of function calls:

```
Queue Q = create_queue(7);
print_queue(Q);
do_something(Q);
print_queue(Q);
```

iii. Propose an alternative solution for `do_something(Q)` function to achieve the same result as the original solution, but using only queues (no stack). Here you are expected to provide a new algorithm for `do_something(Q)` but using only the available Queue operations shown in the given algorithm. **Tip:** Think Recursive!

b) Determine whether the following statements are True or False. If False, provide the correct statement:

i. In an array-based implementation of a List ADT, appending an element to the list and deleting an element from the end of the list are both O(1) time complexity operations.

ii. Deleting an element from a hash table ADT that uses chaining for collision resolution, is a O(1) time complexity operation.

iii. Stack `push()` and `pop()` are O(1) operations, whether the underlying data structure used to implement the stack is an array or a linked-list data structure.

c) Trees

i. Draw an AVL tree that contains the elements (10, 20, 30, 40, 50) which are inserted in this order into the AVL tree. Show all insertion steps and explain balancing steps.

ii. Draw the BST tree after inserting these nodes in sequence (10, 20, 30, 40, 50), then show the resulting Splay tree after searching Node 50. Show all insertion steps and explain Splay rotation steps.

iii. The elements (5, 7, 9, 10, 11, 8) are inserted into a min-heap one by one in this given order. Show the min-heap structure after each insertion, with explanation.

QUESTION 3 [Course Goal 3: Sorting]

a) Bubble Sort

i. Explain Bubble Sort algorithm.

ii. Given an array of integers **[3, 6, 2, 8, 5, 9]**, show the intermediate steps of the array after each iteration of the Bubble Sort algorithm. You are expected to show the state of the array after each iteration that results in a change of the array.

iii. Explain why Bubble Sort is considered inefficient for large datasets, particularly when dealing with worst-case scenarios.

b) Insertion Algorithm

```
Function InsertionSort(Array A, Integer n)

    For i from 1 to n-1
        key = A[i]
        j = i - 1

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

        A[j + 1] = key

    EndFor

EndFunction
```

i. Complete the missing code in blank A for the Insertion sort algorithm above?

ii. Analyze the worst-case case time complexity of Insertion Sort.

iii. Explain why Insertion Sort is a better choice than Quick Sort to handle a dataset of one million nearly sorted records.

c) Consider the following formulation of Quicksort algorithm:

```
Function QuickSort(array, low, high):
    If low < high:
        pivotIndex = choosePivot(array, low, high)
        pivotNewIndex = partition(array, low, high, pivotIndex)
        QuickSort(array, low, pivotNewIndex - 1)
        QuickSort(array, pivotNewIndex + 1, high)
    EndIf
EndFunction
```

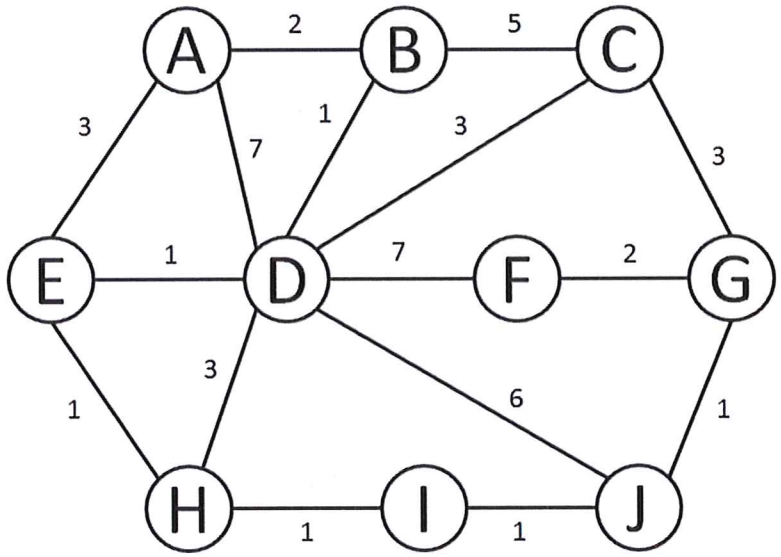
i. Pivot Selection Strategies: Discuss at least two different strategies for implementing `choosePivot(array, low, high)`. Explain how each strategy impacts the performance of QuickSort.

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

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

QUESTION 4 [Course Goal 4: Graphs].

Consider the following graph, where node neighbors are explored in alphabetical order and the search starts at node G.



a) Depth-First Search:

i. Provide a pseudocode of depth-first search (DFS) algorithm.

ii. Specify the order of depth-first search visits in the above graph (DFS traversal).
iii. Provide an application of DFS graph traversal.

b) Breadth-First Search:
i. Provide a pseudocode of breadth-first search (BFS) algorithm.
ii. Specify the order of breadth-first search visits in the above graph (BFS traversal).
iii. Provide an application of BFS graph traversal.

c) Dijkstra:

i. Provide a pseudocode of Dijkstra's algorithm.
ii. Show Dijkstra table that illustrates the order of visits of the above graph.
iii. State the resulting path between nodes G and A and the total cost of that path.

QUESTION 5 [Course Goal 5: Algorithm Design]

a) Algorithm Design

Consider a function *f* where n is an integer ($n \geq 1$):

$f(n)=f(n-1)+f(n-3)$

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

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

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

iii. Describe any relevant observation with respect to efficiency considering the above computation of question ii.

b) Dynamic programming

i. Dynamic programming can be implemented with "bottom up" and "top down" methods. Resolve the above problem (a) with "Bottom Up".

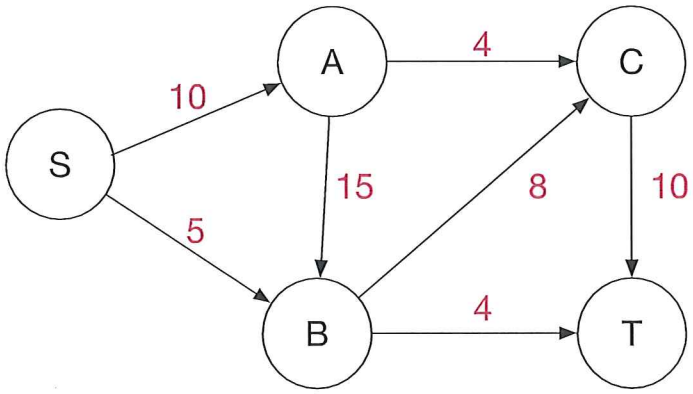
ii. Resolve the above problem (a) with "Top Down".

iii. Discuss comparative tradeoffs between the above two methods.

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.