Analysis of Algorithms (*Five Questions*) Points 100

Q1. Write a recurrence for the time complexity of the following algorithm, and solve it. You may assume n is a power of 2.
Draw the recurrence tree for n=16, and t =15.

```
function f(n, t)
if n ≤ 1:
        print_line(''leaf  \n'')
else
        if t > n then  f(n/2, t)
        else  f(n/2, t)
```

[20]

Q2. Consider the following problem:
INPUT: A set $S = \{(x_i; y_i) \mid 1 <= i <= n\}$ of intervals over the real numbers.
OUTPUT: A pair of distinct intervals from S: $\{(x_k, y_k), (x_m, y_m) \mid k \neq m\}$ such that $(y_m - x_k)$ is the shortest distance amongst all such pairs.
Write a <u>recursive</u> *backtracking algorithm* to solve it. No pruning is necessary. Discuss its time complexity. [20]

Q3. Q1. Write a *dynamic programming* algorithm for computing A(n,1) from the following formula. Analyze the complexity for your algorithm. Draw a template table *A* to explain your algorithm.

A(p, p) are given as input for all *p*, and

A(p, q) = 0, for all $p < q$, for *p* and *q* being integers.

$A(p, q) = \max\{ A(p, k+1) + A(k, q) + 2 \mid \text{for all } k \text{ with } q \le k < p\}$, for all $1 \le q < p \le n$

[20]

Answer briefly (1 or 2 lines each) [20]

Q4a. Can the *Depth first traversal* on a graph produce a spanning tree?

Q4b. Define the *Strongly Connected Component* of a directed graph.

Q4c. What is the main difference between the *Depth first search* on a tree and on a graph.

Q4d. If the *time complexity* of an algorithm is $O(n^2)$ for n input size, what is the largest size of output that the algorithm may produce in terms of n? Explain your answer.

Q4e. On a city street system we would like to know what is the shortest distance between every pair of addresses. Name the specific algorithm for solving this problem.

Answer *true/false* for the following sentences. Explain your answer if you cannot answer as true/false.

Q4f. There are problems which are P-class of problems but not NP-class.

Q4g. The set of P-class problems is a subset of the NP-hard problems.

Q4h. Non-deterministic algorithms are nothing but deterministic backtracking algorithms.

Q4i. In order to prove a problem X to be NP-hard one needs to develop a polynomial transformation from a known NP-hard problem Y to X. So, if Y had a polynomial-time algorithm, then X would also have a polynomial-time algorithm.

Q4j. Suppose, there exists a polynomial transformation from a problem Y to a known P-class problem X. Is Y also a P-class problem?

Q5. *Depth First Search* algorithm for a graph:
Input: graph G=(V, E); *Output*: a DFS spanning tree over G
DFS(node v)
(2a) Write the DFS algorithm to print *pre-order numbering* of the nodes. [20]
(2b) Draw a undirected G=(V={a, b, c, d, e}, E={((a,b), (b,c), (b,e), (b, d), (c,e), (c,d) }. Starting with a call to DFS(a), show your call sequences (i.e., the recursion tree or the traversal of the graph) and your *pre-order numbering* of the nodes. [10]