A **Name:** ______________________________

ADA Final Exam

Short theoretical questions [3.5 points]

[Q 1.] [1 point]

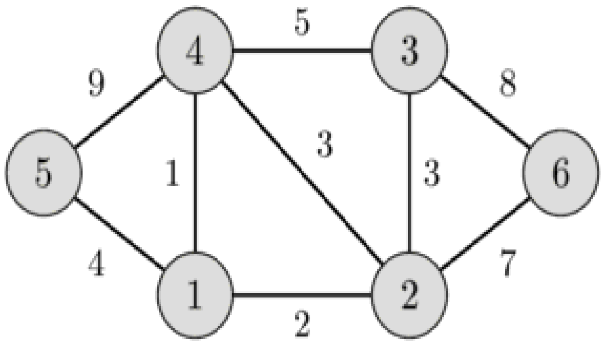
a.) What is a **B-Tree**? Formally define ***the concept*** of B Tree by explaining all the restrictions that apply for its structure. What is its main ***usage and purpose***? [0.30 points]

b.) Depict an **example** of a **B-Tree of mindegree t=3** (Draw one figure representing a valid B-Tree of mindegree t=3 having at least **2 levels**). [0.20 points]

c.) On your chosen B-Tree, give **two examples** to illustrate and **explain** two relevant scenarios that can appear when **inserting** a new key. [0.5 points]

[Q 2.] [0.75 point]

a.) Formally define ***the concept*** of **Minimum Spanning Tree (MST)** of a graph. [0.25 points]



b) Explain **Prim's algorithm with Priority Queue** and illustrate how it works on the graph in the figure. Clearly explain which **datastructures** are needed by the algorithm and **show their content at <u>every step</u> of the algorithm**. You can chose some textual notation, or you can chose to re-draw the graph with some annotations at every step, just make sure that the steps of the algorithm are clearly visible and explained. [0.5 point]

[Q 3.] [0.75 point]

Suppose that in a directed graph there **exists an edge from u to v**. A DFS is performed and following times are recorded for nodes u and v:

- u: discover = 9 , finish = 12
- v: discover = 3, finish = 6

Construct a directed graph that includes the nodes **u** and **v** and has an edge (u,v), such that there **exists a DFS traversal** which assigns to u and v discovery times and finish times values *exactly* as stated before. Also, **draw the corresponding DFS tree**, showing for each node its discovery and finish times (and nodes u and v must have the time values given above!).

[Q4] [1 point]

Identify, explain and correct the problems in the following code.

a.) Make the changes to the code in order to correct the problem [0.25]

b.) Clearly and in detail **explain** why it was a problem and what changes by your correction. Give concrete examples of the incorrect results produced by the function before the correction, and what they become after the correction [0.75]

The code to be analysed is the method `dijkstra` given below. This method takes as arguments a graph, given directly as a simple adjacency matrix, and the start node. It must compute and return an array containing the shortest path values from start to every other node.

```
public class DijkstraMatrix {

    public static int[] dijkstra(int[][] graph, int start) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];

        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        dist[start] = 0;


        for (int i = 0; i < n; i++) {
            int u = -1;
            for (int j = 0; j < n; j++) {
                if (!visited[j] && (u == -1 || dist[j] < dist[u])) {
                    u = j;
                }
            }

            visited[u] = true;

            for (int v = 0; v < n; v++) {
                if (!visited[v] && graph[u][v] != 0 &&
                    graph[u][v] < dist[v]) {
                    dist[v] = graph[u][v];
                }
            }
        }

        return dist;
    }
```

Algorithm design and implementation exercises [5.5 points]

Notes:

- *For all implementation exercises, you are **NOT required to write complete programs**! Focus **only** on implementing **what is required** - **specific operations** and the necessary data structures and variables.*
- *NO POINTS will be awarded if you implement some random operations that are not required!*
- *When appropriate, you should define and use **private helper functions or additional datastructures** to support the implementation of the required operations.*
- *For each of the 3 implemented operations, analyse its **time complexity***
- ***Add comments (free text and/or figures)** to explain and justify your implementation choices. These comments are meant to accompany your implementation code, and cannot serve as a replacement solution instead of the implementation code.*

B-Trees:

a.) Define a B-Tree data structure to hold integer keys. Define classes `Node` and `BTree`, with their class attributes and simple constructors. You do **not** have to implement a method to build the tree! [0.5 point]

b.) In the `BTree` class, define a method `findLevel` that takes as argument an integer value and returns the number of the level where it is found in the the BTree, or -1 otherwise. [1 point]

c.) In the `BTree` class, define a method `isClone` that takes as argument another BTree and returns true if it is a clone of this tree, and false otherwise. Two trees are considered clones if they have exactly the same shape and they also hold the same values. If two trees hold the same values but have different shapes they are not considered clones. Also if two trees have the same shapes but contain different values they are not considered clones. Chose an ***efficient*** implementation for `isClone`. [2 points]

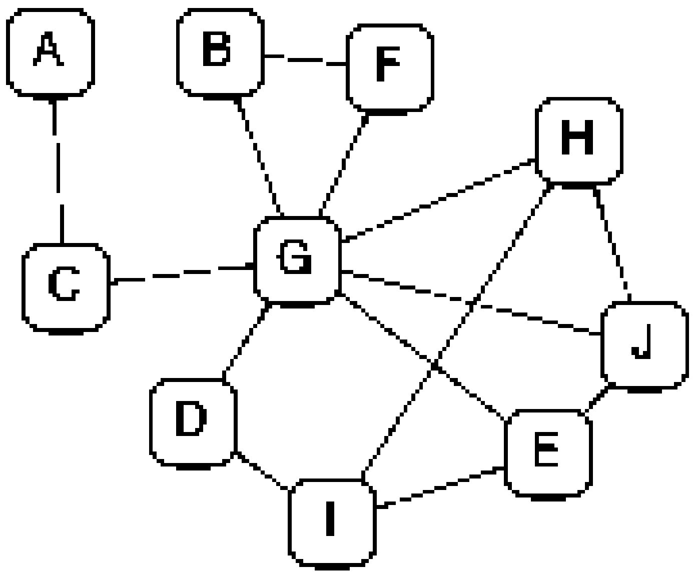
~~Directed~~ Graphs:

a.) Define a UndirectedGraph data structure, based on **adjacency lists** (do **not** use adjacency matrix!) where *nodes are identified by characters*. Define class

`UndirectedGraph`, showing their class attributes and simple constructor. You do **not** have to implement a method to build the graph! [0.5 point]

b.) Make a drawing to show the contents of all the datastructures from point a.) for the graph in the figure. [0.5 point]

c.) In the `UnirectedGraph` class, define a method `checkPath` that takes as argument a String and returns true if it represents a valid path in the graph, and false otherwise. For example, `checkPath("ACGFB")` will return true and `checkPath("ACB")` will return false [1 point]


A
B
F
H
C
G
J
D
E
I