Note:

Since the assignment did not state any particular programming language to use, I decided to work with Javascript for the first two assignments and Python (using Jupyter notebook and pandas) for the third one.

Assignment 1

First, I created an HTML canvas so I could visualize the implemented algorithm working while I was building it. So, if you open the HTML file named “assignment1.html” you will see the canvas with the initial drawing (or testing matrix) with some initial colours.

From the assignment statement I assumed that each pixel within the 2D array would be represented only by and integer (for simplicity) from 0 to 255, which at the same time would represent a colour in the format rgb (x, y, z). So, I created a colour object that you can check in the source code, with initial colours for the canvas background and the linings of the drawing, as well as a function to generate another 253 random colours.

The requested algorithm was done iteratively using an auxiliary stack (array) because even though, it may have been possible to do it recursively, if the matrix is too big and has thousands of pixels, it could crash due to stack overflow.

So, the idea is that I push the initial coordinates given in the parameters (x, y) into the stack and while the stack has some content, I iterate over it. First, I validate the coordinates to check that the passed points actually belong to the matrix. (This has no use if the coordinates are passed to the bucket algorithm through a click event, because every point that I click in the canvas belongs to the matrix. However, if that wasn’t the case, and the initial points are passed randomly I have to perform the check).

Then, I check if the colour of the point being analysed is equal or not to the initial colour. If it is different, it means that there might be a possible shape boundary and therefore it should not change the colour.

Then, if the previous conditions pass, I change the colour of the current pixel to the new colour and I push into the stack, the four pixels surrounding the current one (up, down, right, left) and the loop continues.

This causes to change the colours, pixel by pixel taking into account shape boundaries.

Assignment 2

A and B:

For this task, I assumed that the given graph was either a matrix (2D array) or an adjacency list in the object format for JavaScript “obj {node: [neighbour1, neighbour2, ….]}.

Having said that, I built a JavaScript class named Graph, that takes in the matrix or adjacency list when instantiated. The constructor of this class checks what type of input receives and if it actually is a matrix, it converts it to an adjacency list because for the methods built later it is more practical.

C:

Afterwards, I created a method of the class “hasCycle” to detect cycles within a directed graph. I created three sets named “stack”, “unvisited” and “visited” to keep track of the state of each node.

For this I used a recursive “depth first search” algorithm to traverse through the nodes and moving them between the mentioned sets. Each node being visited get into the “stack” set and their neighbour nodes are analysed. If the neighbours of those nodes happen to already be in the stack set, it means that a back edge was found and the graph has a cycle. In that case, the function returns a “Cycle detected” string.

D:

Assuming that the graph given is an acyclic directed graph, to get the order in which a graph should be traversed in order to satisfy node dependency, I built a sorting function using recursion. For this, I created a “stack” array to push the node traversing order and a “visited” set to add the nodes being visited.

Then I loop through the key nodes in the given adjacency list and if the node is already in the visited set, I pass to the next key. For every node that it is not, I execute the sorting function.

This function adds the current node into the visited set, and loop through its neighbours where it recursively calls the function again.

In case a dead-end node is reached, meaning it has no more neighbours or if a node’s children have been visited, I push them into the stack array to define the order.

I built 6 graphs, including the given in the assignment to test the algorithms. Feel free to manually change the function calls to check them. To view the results, a browser console or Node Js console should be used.

Assignment 3

The Python code, processed data and comments can be found in separate PDF archive called Assignment3.pdf