*(a)  From a space complexity and time complexity perspective, does it make more sense to use an adjacency matrix or an adjacency list in order to represent the intensity graph in part 1? Explain your answer.*

At first, I created an adjacency matrix which converted the 2D image array into a 1D array, then I made a 2D adjacency matrix from that 1D array. Once I finished, I realized that every NxM graph would then be turned in to a (N\*M)x(N\*M) graph. Essentially, we would be squaring the amount of space the original graph contained making the space complexity O(P2). Not to mention, this operation takes O(P2) time to complete.

Instead I created a two 2D arrays size NxM which contain the left (leftEdges) and bottom (bottomEdges). Both graphs are size O(P) and are created and summed in O(P). This essentially works the same as an adjacency list, just represented as a matrix.

Thus, to answer the question, it makes more sense to represent the intensity graph from part 1 as an adjacency list rather than an adjacency matrix. It reduces the time from O(P2) to O(P) and reduces the space complexity by the same amount.

*(b)  Describe the algorithm you implemented in part 2 (both text and pseudocode are fine).*

To summarize what I did: I used Prims algorithm and created a 1D list to hold all the results.

Pseudocode:

Create a 1D array to hold the nodes of the Min Spanning Tree (mst).

Create a 1D array to hold the keys of each node, where the key represents the minimum ‘active’ edge (keys).

Create a 1D array to hold a true/false value if the node has already been visited (isInMST).

Set all the values in isInMST to false.

Set all but the first value of the keysto the maximum value java will allow.

Create a variable, currentNode, to keep track of the current node we are traversing.

Create a 1D array to hold all the weights of the Min Spanning Tree (mstWeights).

For all the pixels in the image

Pick a vertex that is not in the min spanning tree and has the minimum of the keys.

Set this vertex to be the currentNode.

Mark the currentNode as visited in isInMST.

Save the minimum key value in the weight tracker, mstWeights.

Determine the edge weight of all the neighbors and keep track of who the neighbors are.

For each of the neighbors of the currentNode.

If the edge weight from the currentNode to the neighbor is less than the tracked keys value of that node,

Update the keys value to the lower edge weight

Save the currentNode to the Min Spanning Tree (mst).

Sum all the weights of the Min Spanning Tree, mstWeights, and return that number.

*(c)  In Big-O notation, state the runtime complexity of the algorithm you implemented in part 2,*

*in terms of the number of pixels, p, in a given input image.*

O(P2)

This is because I did not use a min-heap. Thus, every iteration of the outside for loop needs to find the min number of the keys.

Had I used a min-heap, I would be able to reduce the time down to O(PlogP).