(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 / part 2?

For G=(V, E), the space complexity of adjacency matrix is O(V*V), the time complexity of adjacency matrix is O(V). The space complexity of adjacency list is O(E), the time complexity of adjacency list is O(V). I use adjacency list to represent the intensity graph since it can save memory space and operates quickly, especially when the graph is sparse.

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

In part 2, I use the greedy algorithm similar to Prim's algorithm to get the minimum spanning tree.

- First, I store adjacent edges of each node in a map, the key of the map is the node number, the value is an edge list which includes start node, end node and weight. For row*col size image, the node number is like this:
  {{0, 1, 2, ..., col-1},
  {col, col+1, ..., 2*col-1},
  .....
  {(row-1)*col, ..., row*col-1}}
- Then initialize the priority queue with node0 as root, and put other nodes in an arraylist called notvisited.
- While the arraylist notvisited is not empty, do the following loop: (a) extract the edge with min weight from the priority queue, if the end node of this edge is inside notvisited, then add the edge to the minimum spanning tree and add its weight to prunedweight; (b) delete its end node from notvisited, use the end node as start node to find new edge in notvisited and add the new edge to priority queue.

(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.

It takes O(p) to create notvisited array and O(4p) to create adjacency list.
The time complexity of adding edges to priority queue is O(E*lg(E)).
The loop of extracting min-weight edge is executed O(p) times in total, and each call of finding the edge takes O(lg(E)) time. The total complexity is O(p*lg(E)).
So the total runtime complexity is O(p + 4p + E*lg(E) + p*lg(E)) = O(5p + 5p*lg(4p)) = O(p*lg(p)) since E<= 4p.