Question 1: Shortest Path Composition

(a) We can prove this using contradiction. Lets first start by assuming that p* is the shortest path from a to b, and has a total cost of x = p1 + p2. However lets assume that p1 is not the shortest path from a to c. Lets call this shorter path from a to c p3. We can now form a new path from a to b such that it uses $p3 \cup p2$, this path will have a new length which we can call y, which is the sum of the lengths of p3 and p2. Since p3 $<$ p1 then $p3 \cup p2 < p1 \cup p2$, which means we have now found a shorter path than p* which is impossible, meaning that p1 must be the shortest path from a to c.

In order to prove the same fact for p2, we can apply the same logic as above without loss of generality. This means we now know that p1 and p2 must be the shortest available paths from a to c and c to b.

Question 2: Best and Worst Cases for DFS

(a) The Best case for DFS is the situation where, the search algorithm directly searches the branch such that it immediately finds the vertex we are searching for. In this case it would take O(1) time to find vertex b from a source a.

(b) In the worst case DFS, may search all other branches of the graph before finally reaching the destination. In this case it is possible to search every other vertex and edge before finally reaching our destination in a total of $O(|V| + |E|)$ time.

Bonus: Pac-Man Heuristic

Algorithm 1: heuristic($food\_remaining\_locs, current\_locs$)

```
   /* Turn coordinates in current loc and food remaining loc into adjacency list    */
 1 G = { } /* initialize adjacency list as empty set                                 */
 2 for coordinate : food_remaining_loc do
 3     for other_coordinate: food_remaining_loc do
           /* calculate the distance between 2 co0rdinates and add it into the adjacency
              list                                                                    */
 4         distance = |coordinate[0] − other_coordinate[0]| + |coordinate[1] − other_coordinate|
           append G[coordinate] with new pair (other_coordinate, distance)
   /* Create the MST between all "occupied vertices"                                */
 5 numEdges = 0 /* The number of edges in the MST                                   */
 6 w = 0 /* The total weight of the tree, which is the heuristic                    */
 7 edges = /* All edge pairs in the graph of pellets and current square in tuple
       (w[u][v], [u][v]), where T[0] is the weight and T[1] is the edge between u,v in
       G                                                                           */
 8 edges = sort(edges) /* Sort in O(mlogn) time using quicksort by their weights from
       smallest to largest                                                         */
 9 for vertex v : G do
10     MAKE − SET(v) /* Make a set for union-find                                   */
11 for edge (u,v), weight W : edges do
12     if FIND-SET(u) != FIND-SET(v) then
           /* Does not create a cycle                                             */
13         w+ = W /* Increment the weight                                         */
14         UNION(u, v) /* Combine the two sets                                    */
15         numEdges + +
           /* Should have n-1 edges                                               */
16         if numEdges == G.size() - 1 then
17             break
18 return w /* Weight of the MST is the heuristic                                  */
```