(1) i) A back edge connects a vertex to its ancestor. Every vertex in the graph is enqueued and therefore dequeued only once because vertices are only enqueued when they are white and then immediately colored gray. Only white vertices are discovered, so a back edge cannot occur because every vertex's ancestors have already been colored black.
BFS produces a single tree since it visits all reachable vertices from the root only once. Therefore there cannot be any forward edges because there are no non tree edges.

ii) BFS finds the shortest path from the source to the vertex. It also produces a tree so if (u,v) is a tree edge then $u.d < v.d$ and $v.d = u.d + 1$

iii) A cross edge can connect siblings but not ancestors so $u.d$ can equal $v.d$ and $v.d$ can equal $u.d + 1$

b) i) In a directed graph, the adjacency list of a vertex that has not yet been scanned can include a white ancestor because in a directed graph not every part of connected vertices is reachable both ways, so it can have back edges. BFS still produces a tree for directed graphs so forward edges cannot exist.

ii) Same reasons as part a). whether directed or undirected if v is discovered after u and (u,v) is a tree edge then $v.d = u.d + 1$

iii) Cross edges can connect siblings so $u.d$ can equal $v.d$ they cannot connect ancestors but can go between vertices in the same tree so $v.d$ can equal $u.d + 1$

iv) A back edge (u,v) connects a vertex to its ancestor BFS finds the shortest path for each vertex and produces a tree so $u.d \geq v.d$, equal on a self loop $v.d = 0$ if v is the root.

(2)
```
for each v∈V // O(V)
  G.Adj[v] : vertices reachable from v. G is input graph
// Adj[v] is a linked list of vertices reachable from v
// Adj[] is an array of linked lists for each v
Q = ∅
Enqueue(Q, s) // s is root, input to algorithm
while Q ≠ ∅
    u = Dequeue(Q)
    u.min = ∞
    for each v∈ G.Adj[u]
        if v.color == White
            v.color = GRAY
            Enqueue(Q, v)
        if u.min > v.label
            u.min = v.label
```

Coloring the vertices ensures each vertex is enqueued only once. Queue operations are $O(1)$ for each vertex which is $O(V)$. Each adjacency list is scanned once and the sum of adjacency lists is $O(E)$. Initializing the adjacency lists is $O(V)$. So total running time is $O(V+E)$

(3) a) If the degree of a vertex is 1 a cycle is not possible in the graph. In general if the degree is odd there will not be a way to leave the vertex without repeating an edge and it is not possible to have only one vertex with an odd degree.
A tour always ends up at a vertex of odd degree with nowhere to go. If the tour starts at a vertex of odd degree it ends up at the odd vertex it is connected to.

If all vertices are even, you can find a unique pair of edges for entering and exiting the vertex each time the vertex is visited.

b) EulerTour(G, s) // G is input graph, s is starting vertex
MaxVisits = (G.Adj[s].size())/2 ; // Maximum number
// of times each vertex should be visited is the
// number of pairs of incident edges.
while(no v in V exceeds MaxVisits)
i = min(v.visits ; v ∈ V)
next = s
for a ∈ G.Adj[next]
if a.visits ≤ i
a.visits++
s = a
break

(4) The running time of Prim's algorithm depends on the min priority queue. Extract-Min depends on Min-Heapify which is logarithmic in the size of the input array which is |V|. It doesn't matter how large the range is, 100 can be compared to 1 just as quickly as 2 can be compared to 1, in O(1). Extract-Min is O(lgV) and it is executed |V| times by the while loop. The for loop within the while loop is executed 2|E| times, for O(E). There is a decrease-key operation in the for loop which takes O(lgV). So the total time is O(VlgV + ElgV) = O(ElgV) for a binary min heap. With fibonacci heaps, extract-min takes O(1) so the running time is O(E + VlgV)

(5) The algorithms of Kruskal and Prim both examine the edges in non decreasing order. This allows for more than one MST if there are ties with edge weights because the algorithms do not care to break ties in a consistent way. If all edge weights are unique then the algorithms examine the edge weights in decreasing order and there is only one permutation, so the MST will be unique.