1a. Two Loops:

"First loop loops over all of the vertices in the graph

Second loop will loop over all vertices connected to the current vertex I'm looking at

This function will loop over all values in the adjacency matrix. It will do this 1 row at a time, as output by G.adj(v)

Two loops: First one from 0 to Second one from … 0 to V.

This is a double loop, so it has quadratic run time. It is O(V^2)." - indirected graph notes

It has V^2 run time from because it still contains redundant information like keeping connections that don't exist etc.

1b. "My Iterable output from G.adj(v) is no longer a row of length V.

But now, all I get is an ArrayList(); that contains only the number of vertices connected to a given vertex.

Or in other words … the degree of the vertex.

The outer loop will run 'V' times for the entire length of the graph

But the inner loop will only run degree(v) times, where 'v' is the given vertex at any run of the first loop.

This loop is … degree(0)+degree(1)+…degree(V).

It is the sum of all degrees in the graph." - indirected graph notes

The sum of degreee would be 2E

Then there is the inner loop which is V

So the time is V+2E

1c. The running time for Directed Graphs Adjacency Matrix Representation is V since now there are no more useless informations that will still be kept

1d. The running time will be Directed Graphs Adjacency List Representation is V+E because outdegree(v) + indegree(w)

2a.

Parent		5	4	2	1	0
Number	0	1	2	3	4	5

2b.

parent		5	0	2	0	0
number	0	1	2	3	4	5

3a.

Parent			0	2	0	0
Number	0	1	2	3	4	5

3b.

Parent			0	2	0	0
Number	0	1	2	3	4	5

4a.

```
DFS(G,v)
        Stack S := {};
        for each vertex u, set visited[u] := false;
        push S, v;
        while (S is not empty) do
          u := pop S;
          if (not visited[u]) then
            visited[u] := true;
            for each unvisited neighbour w of u
              push S, w;
          end if
        end while
      END DFS()
```

4b.

```
BFS
          Set nodes to not visited
          Q = new queue
          Enqueue the inital node
          while(q isn't empty)
                    X = dequeue
                    if( x has not been visited)
                    visited(x) = visted;
                    for( every edge(x,y)
                              if( y is not visited )
                                        equeue(y);
```

6.

```
DFS(G,u){
  visit[u]=1
 for u->v
```

```
if(!visit[v]){
parent[v]=u;
}
else if(parent[v] is not u){
//cycle exists
print(u);

}
```