1.

1)

since $(u, v) \in E' \Leftrightarrow \exists x \in V, (u, x) \wedge (x, v) \in E$ and the graph is given in an adjacency list, we could loop through that list and for each vertex go to all its adacent vertices, and then all the verticies adjacent to those would be made adjacent to the original vertex in G'. sudo code would look like

```
for each v in verticies
  k = new vertex;
  for each i in v.adjacent
    for each j in i.adjacent
      k.adjacent.add(j);
  verticies_prime.add(k);
```

where |v.adjacent| + |i.adjacent| $\leq$ $|E|$ so the max time bound would be $|V| \times |E| = O(mn)$

2)

```
for(var i = 0; i < matrix.length; i++){
  for(var j =0; j < matrix.length; j++){
    if(matrix[i][j]){
      for(var k = 0; k < matrix.length; k++){
        if(matrix[j][k] && k != i){
          new_matrix[i][k] = 1;
        }
      }
    }
  }
}
```

since if(matrix[i][j]) is true exactly $|E|$ times the total run time would be $|V|^2 + |E| \times |V|$ because the inner most loop will run $|E|$ times.

2.

```
        var from = [];

        for(var i = 0; i < list.length; i++){
          from[i] = 0;
        }

        for(var i = 0; i < list.length; i++){
          var adj = list[i];
          for(var j = 0; j < adj.length; j++){
            var adj2 = list[adj[j]];
            if(adj2.includes(i)){
              from[i] = -1;
              from[j] = -1;
            } else{
              if(from[adj[j]] > -1){
                from[adj[j]]++;
              }
            }
          }
        }

        for(var i = 0; i < list.length; i++){
          if(list[i].length == from[i] && 2*from[i] + 1 == list.length){
            return i;
          }
        }
      }
```

my algorithm makes a array of size $|V|$ which stores the amount of verticies with edges pointing at the vertex i where i is the index in the array. once this array is make it loops through and tries to find a vertex where the length of the array of edges going out equals the number of edges coming in which also equals $\frac{|V|-1}{2}$.
We can prove this algorithm is correct directly because if $TO_v \cap FROM_v \neq \emptyset$ then when $i = v$ and $adj[j] = (TO_v \cap FROM_v)(1)$ then then adjacency array for $adj[j]$ must contain i which would exclude vertex i and j from being considered for the center since $from[i] = from[j] = -1$ and the adjacency array for i and j cannot be -1 and once $from[x] = -1, x \in V$ it cannot change. Then a vertex x can only be selected if $|FROM_x| = |TO_x| = \frac{|V|-1}{2}$ because of the last if statement.
Initializing the from array takes $|V|$ time then calculating the values for the from array takes $|V| \times |E|$ time since $|adj| + |adj2| \leq |E|$ and the last loop that checks the values of the $|FROM_v|, |TO_v|$ takes $|V|$ time because it has to check each element. so the total time will be

$$|V| + |V| \times |E| + |V|$$
$$|V|(2 + |E|)$$

3.

1)

suppose there is a cut vertex $v \in V$ that is also a leaf in a DFS search starting on an arbitrary vertex $u \in V$. Since v is a cut vertex that means that V can be split into two sets X, Y that are the set of vertices on each side of the bridge. WLOG we can say $v \in X$. if $u \in X$ then when we arrive at v in the DFS starting at u none of the vertices in Y can have been visited yet by the definition of a bridge vertex. This means that v cannot be a leaf in the DFS tree when $u \in X$ since a leaf in a DFS tree cannot be adjacent to an unvisited vertex when it is reached in the search. If $u \in Y$ and v is a bridge vertex that means that $|X| \geq 2$ because if v is the only vertex in X then removing it would not disconnect the graph. Now when v is reached in the DFS starting at u none of the other verticies in X can have been visited by the definition of a cut vertex which would mean that that v cannot be a leaf in the DFS tree because at least one vertex in X will be its child. this contradicts the assumption that v is a cut vertex in V as well as a leaf in the DFS starting at u proving that v cannot be a leaf in the DFS tree.

2)

my algorithm would be to run a modified DFS on an arbitrary vertex $v \in V$, once i reach the first vertex u that isnt adjacent to any unvisited vertex i would return u. We can prove this is correct directly because u would be a leaf in the DFS tree of G which means it can't be a cut vertex in G by the proof for the last questions. this algorithm must also have a time bound that is less than DFS since it is a DFS that ends at the first leaf. So the time is bounded by $O(m + n)$ since that is the time bound of DFS.