1.) Getafix(G(V,E), D, L): // D is getafix node & L is distance

Init $\forall v \in V$.

BFS(D)

sum = sum of $(\forall v \in V)$.layer (other than D)

reverse (G)

BFS(D)

sum += sum of $(\forall v \in V)$.layer (other than D)

return sum * L

BFS(D) gives us the shortest distance from D for each V

Reversing G then BFS(D) would give us the shortest Distance to D for each V

Sum up all those distances since Getafix has to go to each house and back home each time

Times length given

Init is $O(|V|)$

Summation happens in $O(|V|)$

BFS in $O(|V|+|E|)$

and reverse in $O(|E|)$

so $O(|V|+|E|)$ is greatest among them

$\rightarrow \in O(|V|+|E|)$

2.)

```
Strong(G(V,E)):
    reverse(G)
    DFSExplore(G)
    for ∀v∈V
        x=v if v.et > x.et
    Init ∀v∈V
    DFS(x)
    return (x.et == 2|V|-1)
```

|V|-1 Strong vertex means only one BSCC

Highest end time vertex is in BSCC of G^T after BFS Explore (G)

I did BFSExplore(G^T) to find BSCC of G

Reverse $\in O(|E|)$
Init $\in O(|V|)$
DFS & Explore $\in O(|V|+|E|)$
For loop $\in O(|V|)$

$\rightarrow O(|V|+|E|)$

If only one BSCC then it should be reachable for all other nodes in G or reach all nodes in G^T (which is what I did)

x.et should be 2|V|-1 if all nodes reached

3.)

```
Cycle(G(V,E)):

    Init ∀v ∈ V    visited = false  parent = -1

    for v ∈ V
        if v not visited
            if DFS(v) == true
                return true

    return false

    DFS(v)
        v.visited = true
        for all u ∈ N(v)
            if u not visited
                u.parent = v
                return DFS(u)
            else if v.parent is not u
                return true
            else
                return false
```

Modified DFS Explore

checks if an already visited neighbor is not its parent

if there exists such a neighbor then this node connects to an ancestor in the DFS tree which creates a simple cycle starting at that ancestor

As this is a modified DFS Explore only increase constant time in loop

So still

$\in O(|V| + |E|)$

4c)

```
One-Way(G(V,E)):
    scc_list = Kosaraju's(G)
    if scc_list length != 1
        return true
    SG(SV,SE) = toGraph(scc_list)
    for ∀v ∈ SV
        if |N(v)| or |N^T(v)| > 1
            return false
        if |N^T(v)| == 0
            x = v
    DFS(x)
    return (x.et == 2|SV| - 1)
```

```
toGraph(scc_list):
    New graph
    add vertex for each scc
    for each scc
        for each v in scc
            for each u ∈ N(v)
                add edge from
                u's scc to v's scc
    return Graph
```

Collapse graph to DAG of SCCs or one SCC

DAG would be one way iff one topological ordering (i.e. straight line) so only at most 1 incoming & outgoing for each vertex

I do DFS on vertex with no incoming edges which should be first/top of topological ordering

if it doesn't reach all nodes then sub graphs exist

Kosaraju's $\in O(|V|+|E|)$

toGraph would go over every edge & possibly every node $\in O(|V|+|E|)$

for ∀v ∈ SV worst case is that each node was SCC alone so $\in O(|V|)$

DFS $\in O(|V|+|E|)$ if graph was same

$\rightarrow O(|V|+|E|)$

5.) 2-colorable(G(V,E), C1, C2):

BFS Explore(G)

For ∀v ∈ V:

for u ∈ N(v)
if v.layer == ulayer
return false

For ∀v ∈ V:

c(v) = if v.layer is even C1 else C2

return true

I do bipartite test since this is only 2 colorable by checking if no node on a layer is connected to that layer.

Then, I split colors by marking first color for even layers and second for odd layers

BFS Explore ∈ O(|V|+|E|)

First for loop only goes over each edge twice ∈ O(|E|)

Second loop only over each vertex O(|V|)

→ O(|V|+|E|)