Design and implement an algorithm to find a cycle (just one cycle) in an undirected graph.

Problem 1
[10 pts] Design a correct algorithm and show it in pseudo-code

Soln:

Algorithm 1 Detect Cycle Depth First Search

```
procedure IsCyclic(G)
    for each vertex v in G do
        visited[u] ← false
    for each vertex v in G do
        if v is not visited then
            if DFSUtil(G, v, −1, visited) = true then
                return true
    return false

procedure DFSUtil(G, v, parent, visited)
    visited[v] ← true
    for each neighboring vertex u of v do
        if u is not visited then
            if DFSUtil(G, u, v, visited) = true then
                return true
        else
            if u ≠ parent then                    ▷ A backedge
                return true
    return false
```

Problem 2
[10 pts] Provide proof of the algorithm's correctness.

Soln: To prove the correctness of the algorithm above that uses DepthFirstSearch, let us first prove the following:
Lemma1: A directed graph contains a cycle if and only if DepthFirstSearch on that graph classifies some edges as back edges.

Proof. Suppose G has a back edge (v,u). Then v is a descendant of u in the DFS forest. Therefore, a cycle from u to v in G can be obtained by going from u to v via tree edges and then going from v to u via (v,u). Suppose G has a cycle C. Let u be the first vertex in C to be discovered. Since there is a path from u to all other vertices in C and those vertices are white when u is discovered, all those vertices will become descendants of u by the white path theorem. Since u is in a cycle C, there is at least one vertex v in C such that (v,u) is in G. Since v is a descendant of u, (v,u) will be classified as a back edge. Therefore, if G is cyclic, it has a back edge. ☐

The algorithm *DetectCycleDepthFirstSearch* returns true as soon as it encounters a back edge. As we know that using *Lemma*1 a graph has a cycle if it has a back edge then the algorithm always finds a cycle if there is one.

Problem 3
[10 pts] Find and prove the algorithm's running time.

Soln: The time complexity of the algorithm above is O(V) where V is the number of nodes.

- A tree contains V-1 edges and a cycle should contain atleast V edges.
- The above algorithm visits every node once.
- If there is a cycle, the above algorithm visits atmost V nodes before discovering a back edge and terminates, hence V operations.
- If there is no cycle it traverses all nodes only once before terminating, also V operations.
- With running time linearly dependent on the size of vertices or the number of operations, the time complexity hence would be O(n)

Problem 4
[**20 pts**] Implement the algorithm in a compiled language

1. Write a graph generator.
2. Write test code to validate that the algorithm finds cycles.
3. Test the algorithm for increasing graph sizes.
4. Plot the running time as a function of size to verify that the asymptotic complexity in step 3 matches experiments.

Soln: Running time as a function of size to verify the asymptotic complexity.

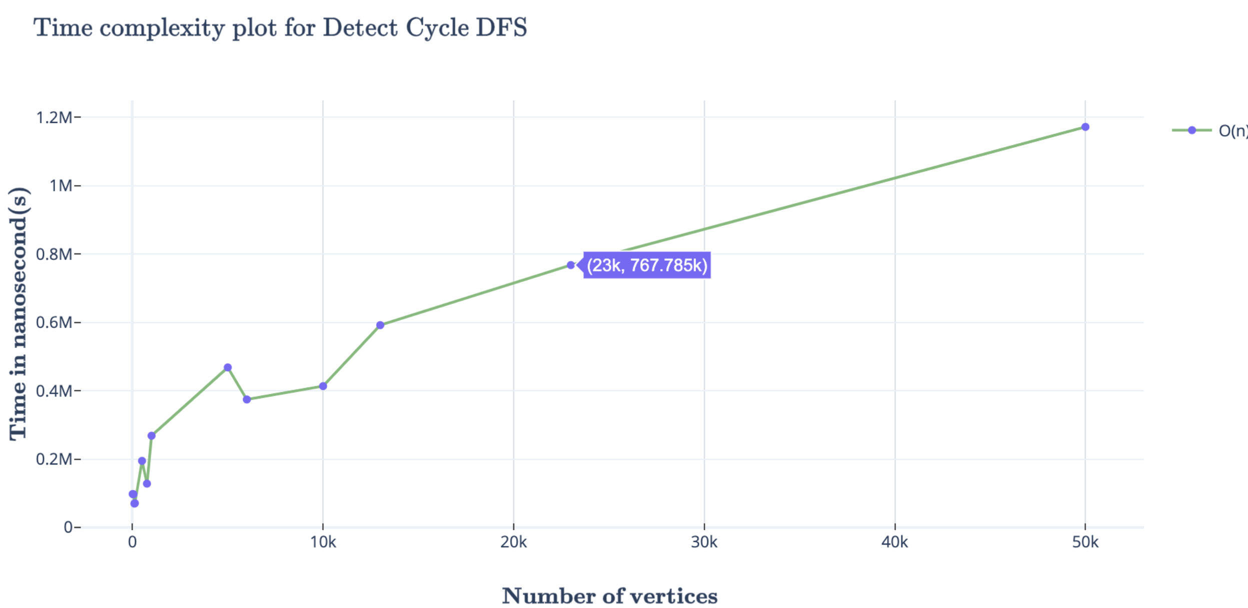

Figure 1: Detect Cycle Depth First Search performance