Consider undirected graphs that have n nodes and at most n+8 edges. For these graphs, design an efficient algorithm that finds the minimum spanning tree.

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

Soln:

Algorithm 1 Find minimum spanning tree Depth First Search

```
procedure IsMST($G$)
    for each vertex $v$ in $G$ do
        $visited[u] \leftarrow false$
    $maxEdge \leftarrow randomedgefrom\ \ G$
    for each vertex $v$ in $G$ do
        if $v$ is not visited then
            if $DFSUtil(G, v, -1, visited, maxEdge) = true$ then
                return $true$
    return $false$

procedure DFSUtil($G, v, parent, visited, maxEdge$)
    $visited[v] \leftarrow true$
    for each neighboring vertex $u$ of $v$ do
        $maxEdge \leftarrow maximumOf(maxEdge, edge(u, v))$
        if $u$ is not visited then
            if $DFSUtil(G, u, v, visited, maxEdge) = true$ then
                return $true$
        else
            if $u \neq parent$ then                         ▷ A backedge
                $G \leftarrow G - maxEdge$
                return $true$
    return $false$
```

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

Soln: The above algorithm's core logic behind finding a minimum spanning tree is by eliminating the maximum edges. In the given case there are 8 more edges than vertices. Which implies if vertices are say V = v then edge count is E = v+8.

The correctness of the algorithm can be proved by the method of proof by contradiction.

Note: The above algorithm has to be run 9 times in order to remove the maximum weighted edges to get a minimum spanning tree. This is because a tree can only have $V - 1$ edges, which implies in the current case that 9 edges from V+8 edges have to be removed in order to obtain a tree.

Proof. : Assume the above algorithm, after running for stated number of iterations on a graph G does not produce a minimum spanning tree. This means that the graph that is obtained at the end of running the algorithm for so many iterations has at least one edge heavier than what an ideal minimum spanning tree would arise out of G. Since the algorithm always eliminates the maximum edge it encounters during a cycle, at every iteration the edge that is maximum so far in the graph is removed. This can be easily proved by understanding that DepthFirstSearch always detects cycle after it has visited every node in a connected component and therefore all edges that are not part of cycle are visited at least once. Thus after 9 iterations the above algorithm removes the top 9 weighted edges, making the graph a tree with least possible weight, which essentially is a minimum spanning tree. This contradicts our assumption. Therefore it can be proved that the algorithm $FindMinimumSpanningTreeDepthFirstSearch$ with given preconditions on V and E always produces a minimum spanning tree after E-V+1 iterations. □

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.

- We have a precondition that the maximum number of edges that the graph can have is V + 8.
- A tree contains V-1 edges and a cycle should contain atleast V edges.
- The above algorithm visits every node once.
- To find the minimum spanning tree, the above algorithm visits atmost V nodes before discovering a back edge and finally discarding the maximum weight edge.
- However, since there are 9 more edges than required number of edges that can satisfy the condition of a tree(V-1 edges). This algorithm has to be run for a maximum of 9 times. Resulting in the following time complexity: $9 \times O(n)$
- Why order of O(n)? because the algorithm is similar to the first algorithm presented for detecting a cycle. Therefore a constant times O(n)
- $c \times O(n) = O(n)$.
- If there is no cycle it traverses all nodes only once before terminating, also 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 a spanning tree.
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.

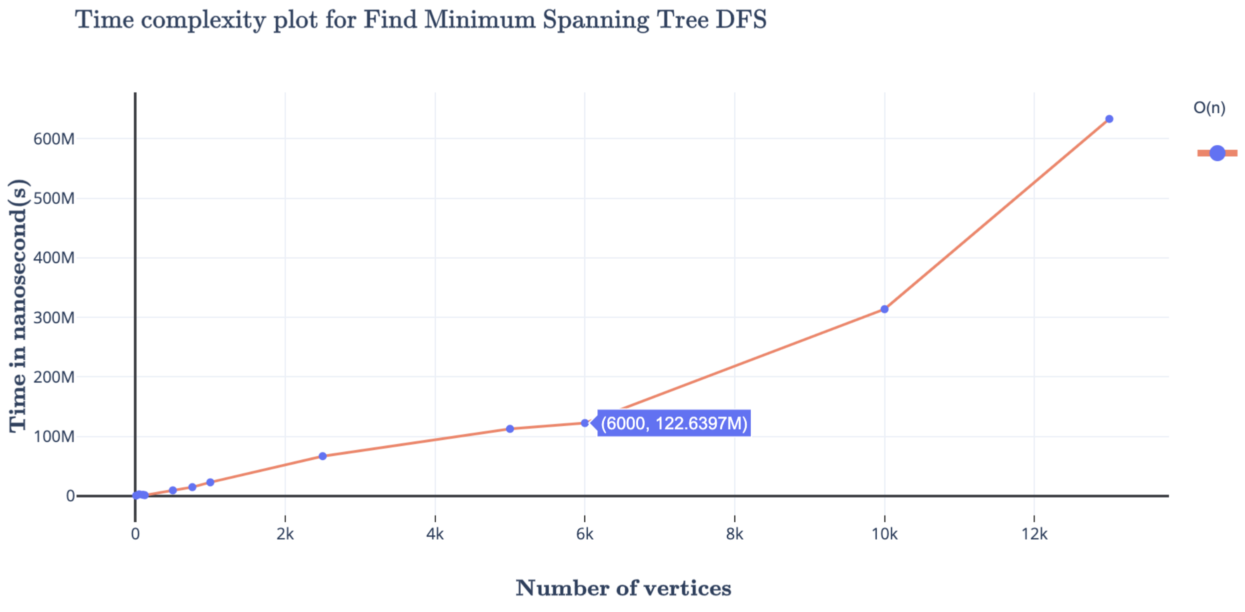

Figure 1: Find Minimum Spanning Tree Depth First Search performance