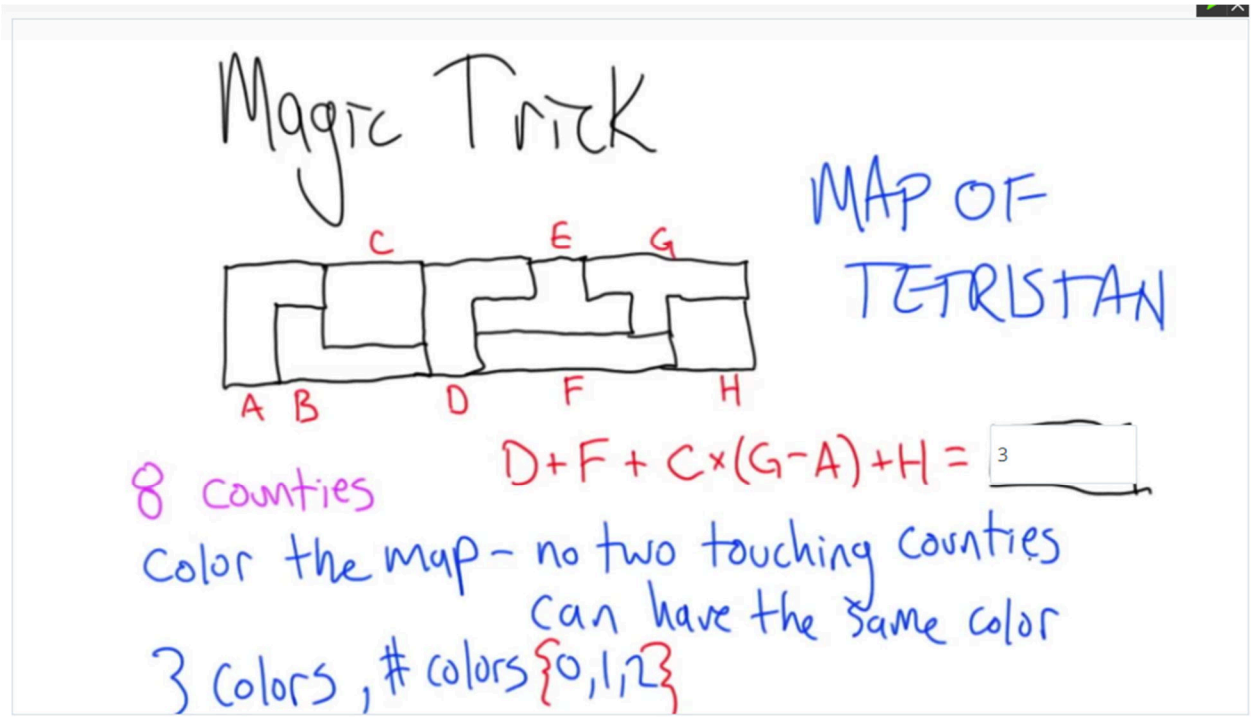
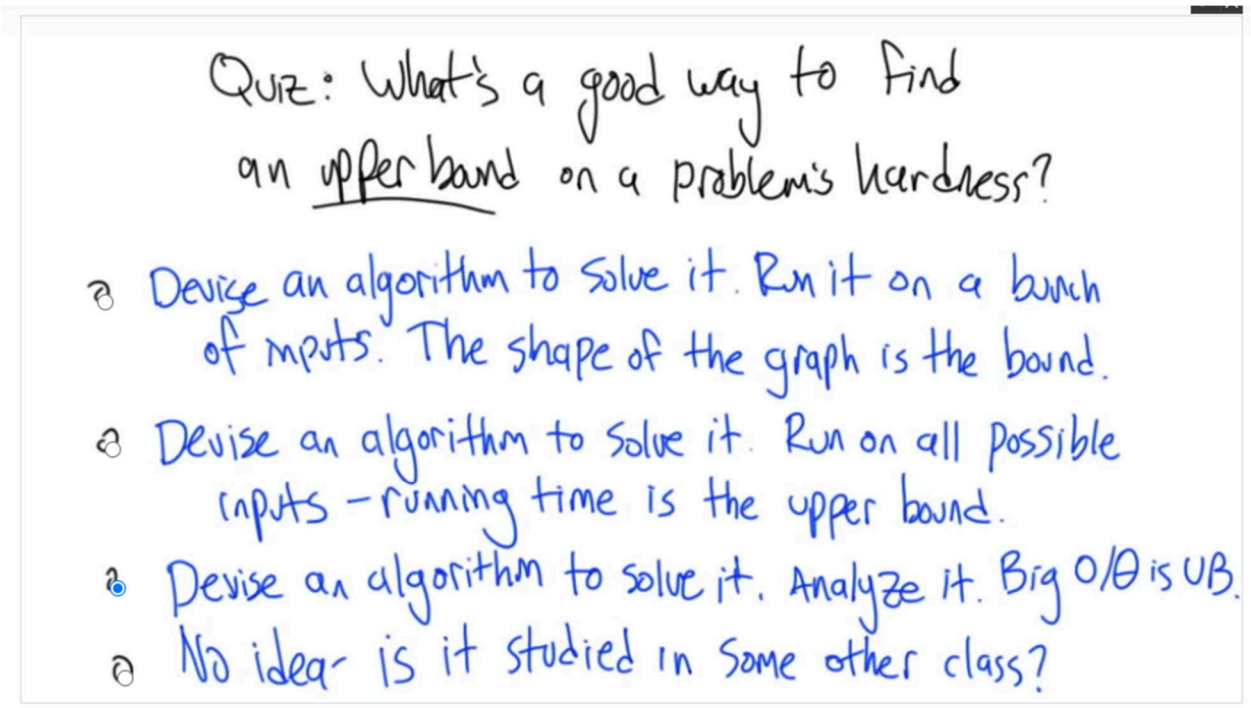
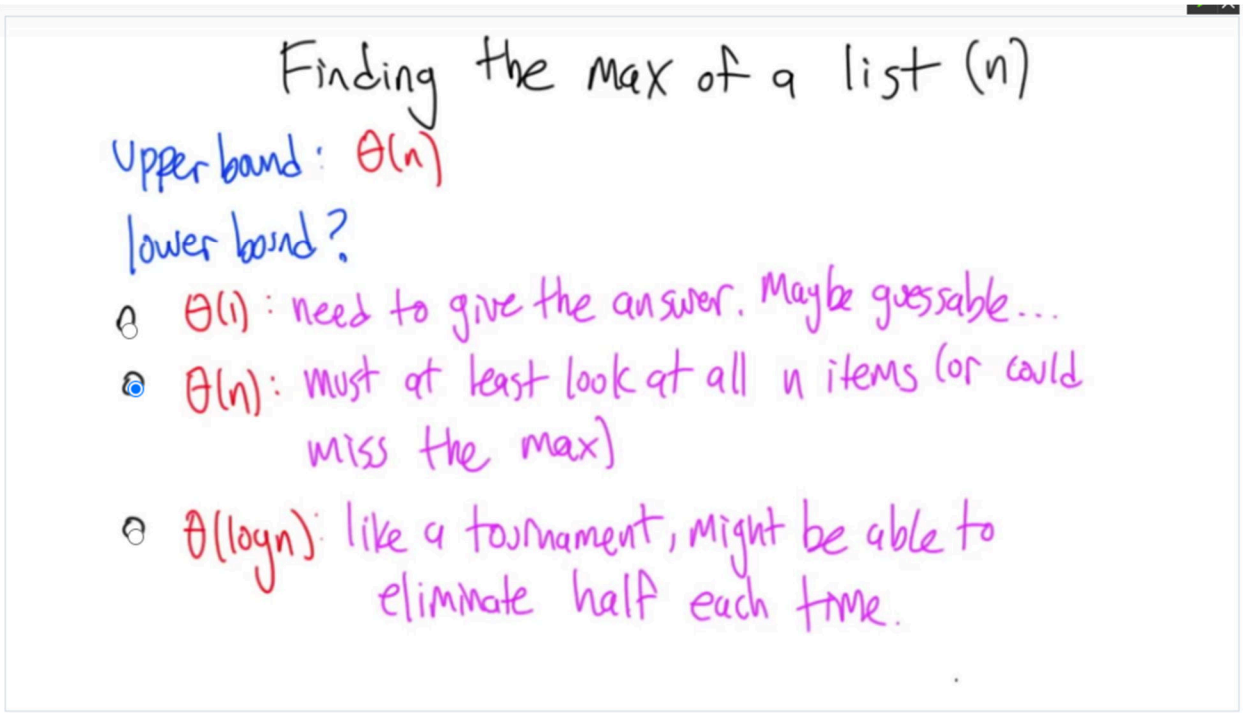
Tetristan



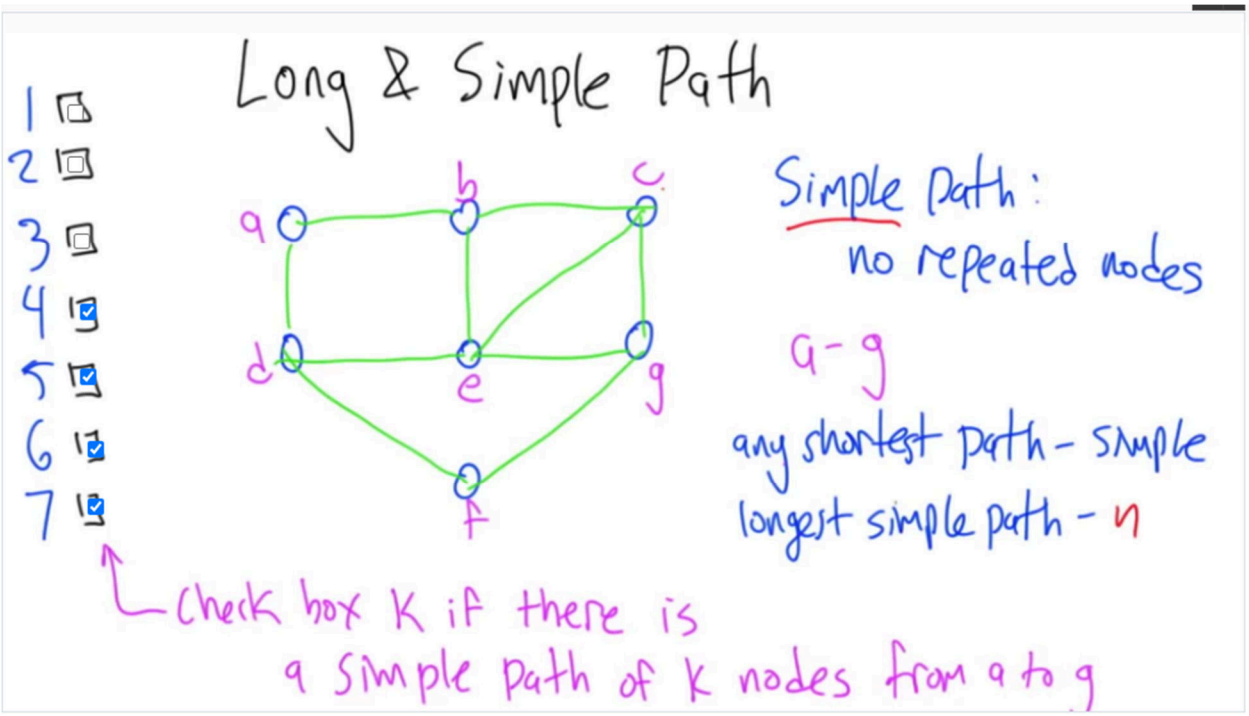
Degrees of Hardness



Lower Bound on Complexity



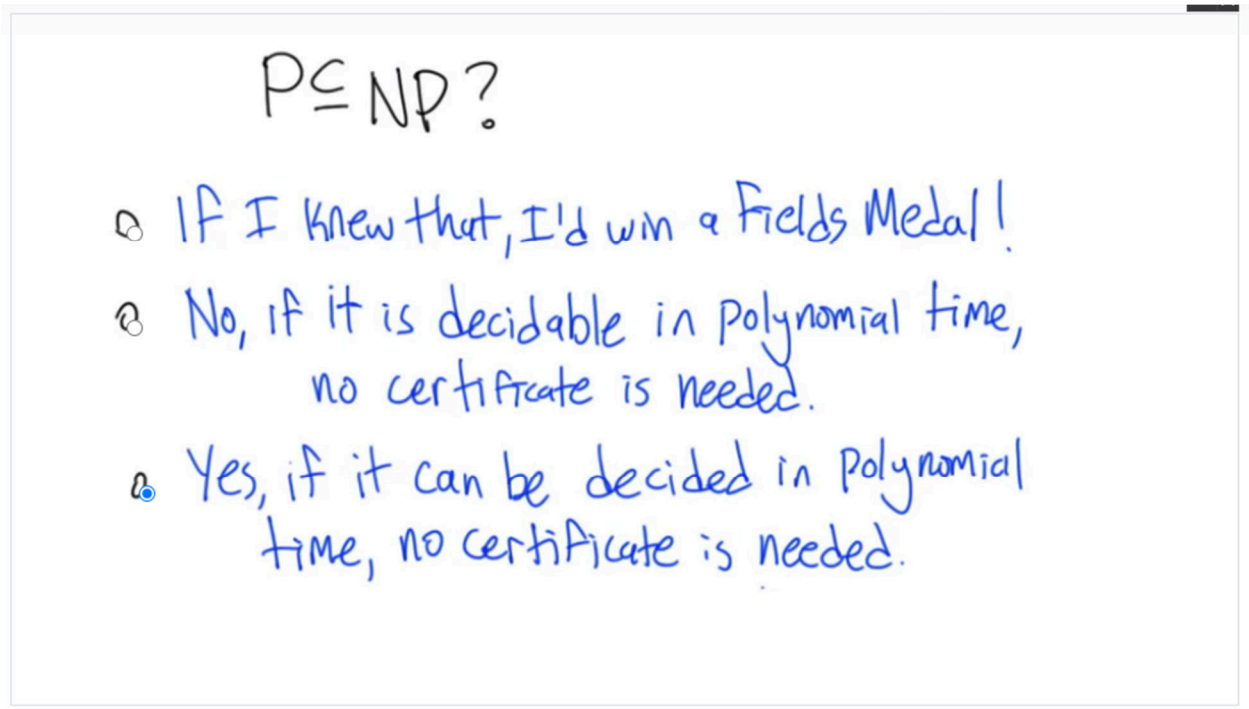
Longest Simple Path



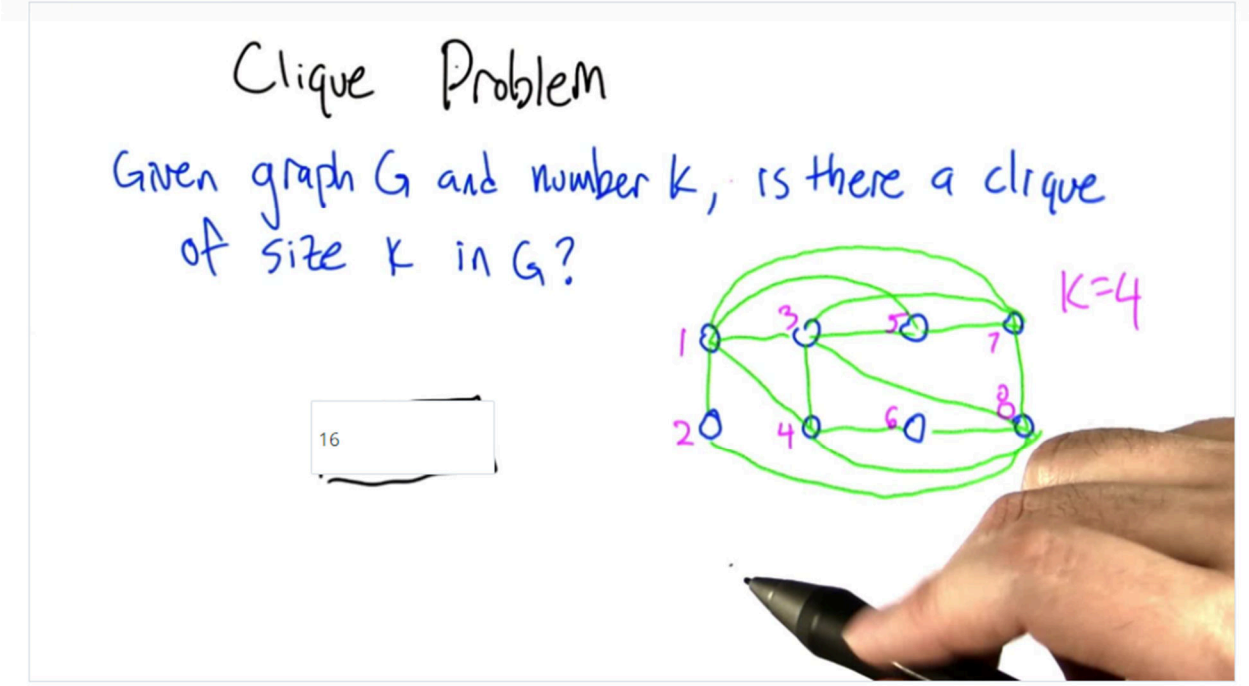
Reduction: Long and Simple Path

```
def long_and_simple_path(G,u,v,l):
    """
    G: Graph
    u: starting node
    v: ending node
    l: minimum length of path
    """
    if not long_and_simple_decision(G,u,v,l):
        return False
    for node1 in G:
        neighbors = G[node1].keys()
        for node2 in neighbors:
            G = break_link(G, node1, node2)
            if not long_and_simple_decision(G,u,v,l):
                G = make_link(G, node1, node2)
    path = [u]
    node = u
    next_node = (G[node].keys())[0]
    while next_node != v:
        path.append(next_node)
        next_next_0 = (G[next_node].keys())[0]
        next_next_1 = (G[next_node].keys())[1]
        if next_next_0 == node:
            node, next_node = next_node, next_next_1
        else:
            node, next_node = next_node, next_next_0
    return path + [v]
```

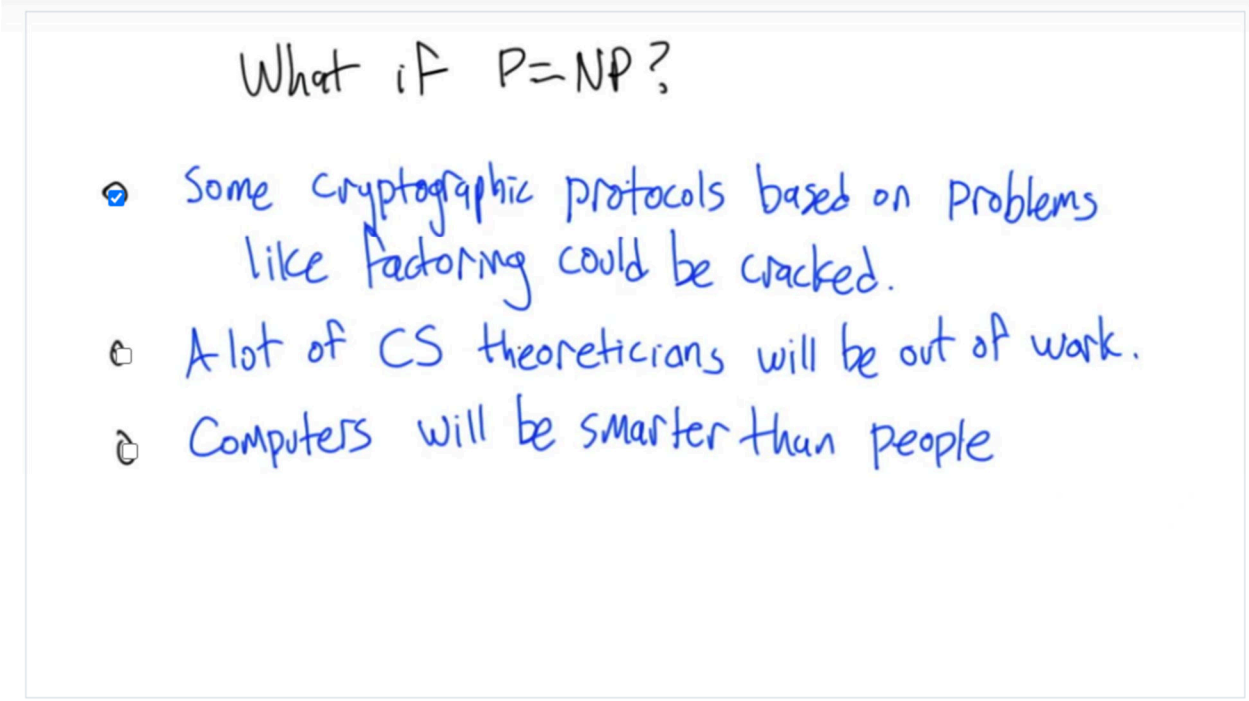
Accepting Certificate



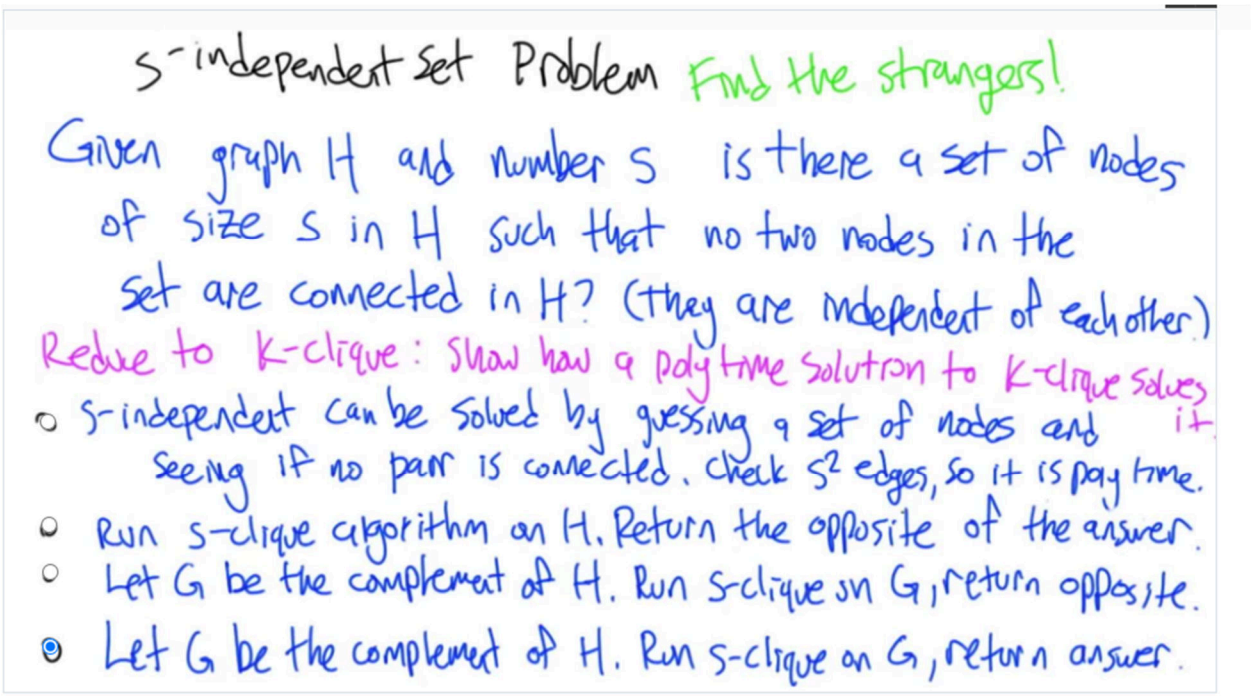
Clique Problem



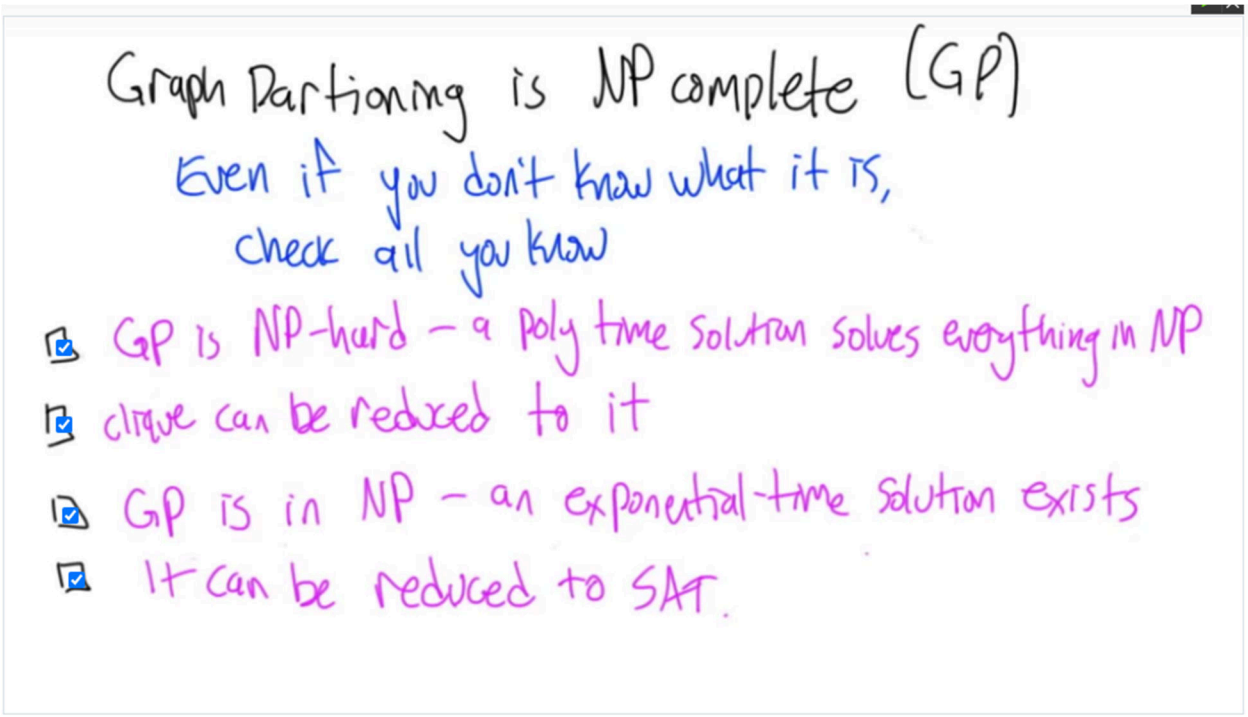
P=NP?



Find the Strangers



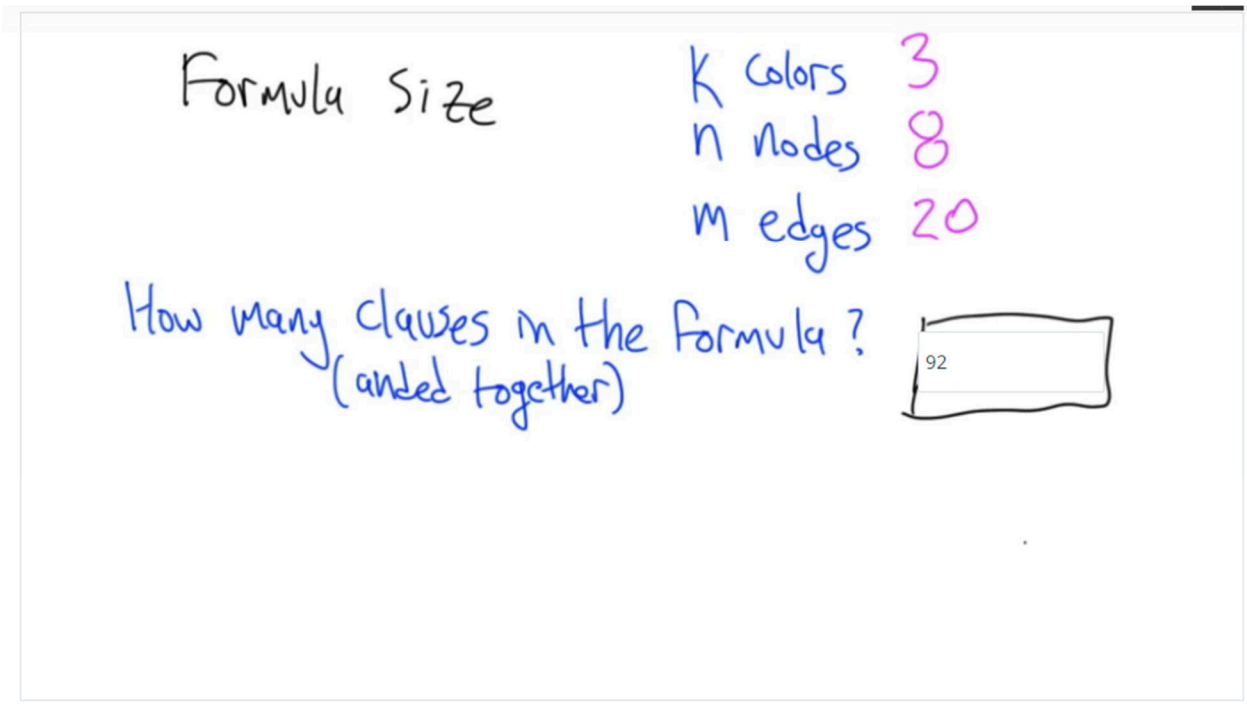
NP-Completeness



Solving 3-Colorability

```
def verify(G, cert, k):
    if len(cert) != len(G):
        return False
    for node in cert:
        if cert[node] not in range(k):
            return False
        for neighbor in G[node]:
            if cert[node] == cert[neighbor]:
                return False
    return True
```

Generating a Formula



4-Colorabiity