Problem 6-1.

```
def ComputeStrength(V, E, k, s):
    d = [0 for i in range(|V|)]
    d[s] = 1
    for i in range(k):
        for e in E:
            d[e.end] = max(d[e.start]*e.weight, d[e.end])
    return d
```

Correct:

```
def ComputeStrength(V, E, k, s):
    d = [∞ for i in range(|V|)]
    d[s] = 0
    for i in range(k):
        T = {}
        for e in E:
            if(d[e.start] + e.weight < d[e.end]):
                T[e.end] = d[e.start] + e.weight
        for v in T:
            d[v] = T[v]
    return d
```

Reason:
For some sequence of edges, the length of path may exceed k. Updating one level each loop guarantees the length.

Problem 6-2.

a: We abstract this problem into a graph. A library is a vertex and and Edge(u to v, directional) means library U is library V's dependency. Then we do topological sort to this graph (deep first traverse, add a vertex to the result list then this vertex finishes, then reverse the list).
Correct: Edge(u to v, directional) means library V is library U's dependency and do not reverse the result list.

b:

```
def ComputeOrder(P, D, E):
    res = []
    for v in D:
        DFS_visit(P, v, E, res)
    res.reverse() #wrong and remove this
    return res
```

```
def DFS_visit(P, v, E, res):
    v.visit = True
    for e in E[v]:
        if e.end.visit is None or !e.end.visit:
            DFS_visit(e.end, E, res)
    if v not in P:
        res.append(v)
```

Correct:
If v is installed, the dependencies are installed too. So we don't have to visit the vertex followed. Change "if e.end.visit is None or !e.end.visit:" into "if (e.end.visit is None or !e.end.visit) and v not in P:"