1. Interval scheduling.

The complexity of the problem of Interval scheduling is $O(n \log n)$

a) Interval Scheduling is in P.

TRUE, The optimal version of Interval Scheduling can be solved in polynomial time, therefore the decision version is also in P.

b) Interval Scheduling is in NP.

TRUE, because every problem in P are also in NP

$P \subseteq NP$.

c) Interval Scheduling is NP-complete

Unknown. If Interval Scheduling is NP-complete, every other problem in NP can be reduced into Interval Scheduling in polynomial time. And this is unknown.

d) Interval Scheduling $\leq_p$ Vertex Cover.

True. Because vertex cover is NP-complete and the definition of NP-complete tells every problem in NP can be reduced to it.

e) Independent Set $\leq_p$ Interval Scheduling Unknown.

This depends on P=NP, which is unknown.

Independent Set is NP-complete. If Interval Scheduling is NP-complete, this will be true. But it is undecided whether Interval Scheduling is NP-complete.

2. Vertex Cover on Trees.

(a)

The algorithm :

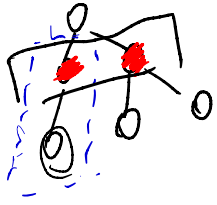
graph $G=(V,E)$, S (Independent Set)

① we find a random leaf v.

② we put the parent of node v into S.

③ we delete u, v and all the edges incident to them.

④ repeat the procedure until there are no edges left.



Independent set.

(b)

To find a maximum independent set of a Tree T

Root the tree at node r

For all nodes u of T in post-order:

If u is a leaf

Mout[u]=0; Min[u]=1;

Else

$$M_{out}[u] = \sum_{v \in children(u)} \max\left(M_{out}[v], M_{in}[v]\right)$$

$$M_{in}[u] = \left(\sum_{v \in children(u)} M_{out}[v]\right) + 1$$

Endif

Endfor

Return $\max(M_{out}[r], M_{in}[r])$

---

① post-order needs dfs, $O(m+n)$.

② During the dynamic programming, we will go through all nodes two times, one is for the traversal, another is serving as other's child.

go through all edge one time. To finish the traversal, the time complexity is $O(m+n)$.

So the total time complexity is $O(m+n)$.

(c) No, it is not contradiction. Althought the question of indepedent set is NP-complete problem, but this question considers only the input of tree instead of graph. Tree is a special graph.

3. The Path Selection Problem.

a) ① prove NP.

The path selection problem is in NP because given a set of k out of c paths, we can check whether two paths have common nodes in polynomial time.

② The path selection problem is NP-complete because other NP-complete problems reduce to it in polynomial time.

Because the set packing problem is NP-complete, we can reduce set packing to Path Selection Problem.

How to reduce set packing to path selection Problem?

construct a graph with nodes corresponding to the elements in U. we make every subset S_i $i=1 \cdots c$ and turn it to a path. Then set packing will become Path Selection Problem.

b) This is a decision problem and belongs to the packing problems.

4. The Traveling Salesman Problem.

a) given a graph $G(V,E)$ with $n=|V|$ nodes and a number k. Each edge (from node i to node j) has a distance d_{ij}. Is there a tour with length at most k?

This is NP problem, we can efficiently check whether a path meets the requirement. As long as, we check whether everynode's appearance time is only once. and add up the route to compare with k.

b) The brute force appoach would consider all the possible permutations of n cities. The complexity of it is $O(n!)$

(c) For a subset of nodes $S \subset \{1,2,\cdots n\}$ we assume S include $1,2\cdots j$ node. let $C(S,j)$ be the shortest distance from 1 to j (1 is starting node) and each node appears once.

we define $C(S, 1) = \infty$

The formula is

$$C(S, j) = \min_{i \in S, i \neq j} C(S - \{j\}, i) + d_{ij}.$$

i is the nodes that are reachable to j.

Running time:

Because the subproblem is almost $2^n \cdot n$,

to deal with one subproblem, we need spare $O(n)$

to find the minimal edge. The running time is

$O(n^2 \cdot 2n)$. This is NP-hard problem.