Problem type 1:

Let's say we have the following two problems:

(See variants below)

As you can see, they are both the same problem but one is a decision version of the problem and the other is a optimization version of the same problem. We have a block-box algorithm that solves the decision version in polynomial time. Using this black box program, describe an algorithm that solves the optimization version of this problem.

Does this algorithm demonstrate Problem$OPT \implies$ ProblemDEC, or Problem$DEC \implies$ ProblemOPT? (select one and draw a box around it on your test sheet).

a. **BYB/BYE**

Independent Set Decision: (**IndSetDec(G, k)**)

- Input: A undirected graph G and integer k
- Output: True if G has a independent set of size $\geq k$, False otherwise

Independent Set Optimization: (**IndSetOpt(G, k)**)

- Input: A undirected graph G
- Output: The *size* of the largest independent set in G

Solution: Simply iterate on k from n down to 1.

```
IndSetOpt(G)
    for k = |V| to 1
        if IndSetDec(G, k)
            return k
```

This reduction shows IndSetOpt $\implies$ IndSetDec ■

b. **BYA/BYH**

Clique Decision: (**CliqueDec(G, k)**)

- Input: A undirected graph G and integer k
- Output: True if G has a clique of size $\geq k$, False otherwise

Clique Optimization: (**CliqueOpt(G, k)**)

- Input: A undirected graph G
- Output: The *size* of the largest clique in G

Solution: Simply iterate on k from n down to 1.

```
CliqueOpt(G)
    for k = |V| to 1
        if CliqueDec(G, k)
            return k
```

This reduction shows CliqueOpt $\implies$ CliqueDec ■

c. **BYC/BYF**

Traveling Salesman Decision: (**TSDec(G, k)**)

- INPUT: A undirected weighted, all-positive graph G and integer k
- OUTPUT: True if there exists a path that visits every vertex exactly once, ends at the vertex it started at and costs $\leq k$. False otherwise

Traveling Salesman Optimization: (**TSOpt(G, k)**)

- INPUT: A undirected weighted, all-positive graph G
- OUTPUT: The *weight* of the minimum cycle in G that visits every vertex exactly once. (-1 if one doesn't exist)

Solution: Simply iterate on all possible weights of the TSP tour. The max possible tour weight would be approximately the maximum edge weight multiplied by $n-1$ (since the tour must have n edges).

TSOpt(G)
    for $k = 0$ to $|V| - 1 \times \ell(E)$
        if TSDec(G, k)
            return k
    return -1

This reduction shows TSOPT $\implies$ TSDEC. ■

d. **BYD/BYG**

kColor Decision: (**kColorDec(G, k)**)

- INPUT: A undirected graph G and integer k
- OUTPUT: True the vertices in G can be colored with k colors such that no two adjacent vertices share the same color, False otherwise

kColor Optimization: (**kColorOpt(G, k)**)

- INPUT: A undirected graph G
- OUTPUT: The *minimum* number of colors needed to color the vertices in G such that no two adjacent vertices share the same color.

Solution: We can assume that the maximum number of colors we'd need is n. So, simply iterate on k from n down to 1.

kColorOpt(G)
    for $k = 1$ to $|V|$
        if kColorDec(G, k)
            return k

This reduction shows KCOLOROPT $\implies$ KCOLORDEC ■