Q3 [5 points] Does this algorithm compute a constant-factor approximation to the largest independent set of an input graph? You may require the fact that if V’ is a vertex cover of G=<V,E> then V-V’ (vertices not in V’) form an independent set, and vice versa.

```
def approxIS(G=<V,E>):
T = minimum spanning tree of G
V’ = internal nodes of T
return V-V’ // return the complement of V’
```

Ans:
No. Consider this infinite family of graphs: for any n, Gn is a line/path-graph with n nodes and edges only between v1-v2-v3... -vn. For such graphs, T is the same as G, and so approxIS will return 1 node. However, the largest independent set consists of alternating nodes whose number is at least n/2. Hence, there is no constant c such that |approxIS| >= |OptIS|/c.
** Q4 [5 points] The SMALLFACTOR problem takes as input a number n in binary and returns true if n has some factor f that is less than $\lfloor lg(n) \rfloor$, else returns false.
Is SMALLFACTOR a P problem, an NP problem, an NP-hard problem, an NP complete problem? Explain briefly.

(b) What is the absolute approximation ratio of this algorithm? Explain.
Ans: [2 points] 1 is the absolute approximation ratio.
If the graph has no edge or is bipartite, then OPT=APPROX.
For all other planar graphs, OPT is 3 or 4, and APPROX=4.
So for all cases OPT <= APPROX <= OPT+1.

Consider this algorithm that tries to determine the chromatic number of a planar graph using the fact that every planar graph can be coloured using at most 4 colours (The story behind the proof of this fact is worth reading later.).

```
def PlanarChromatic(planar graph G):
if G has no edge: return 1
else if G is bipartite: return 2
else return 4
```

(a) What is the relative approximation ratio of this algorithm? Explain.
Ans: [3 points] 4/3 is the relative approximation ratio.
If the graph has no edge or is bipartite, then OPT=APPROX.
For all other planar graphs, OPT is 3 or 4, and APPROX=4.
So for all cases, OPT <= APPROX <= OPT*(4/3).

Q)

A programmer incorrectly implemented the Floyd-Warshall algorithm with the following code:

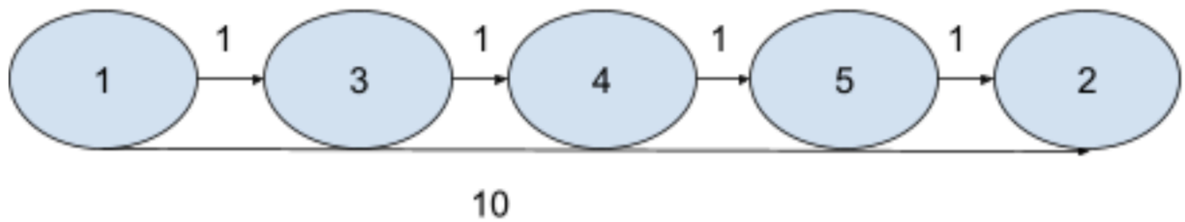
```
python
function FloydWarshall(int[N,N] c) {
   int[N,N] a = copy(c);
   for (i = 1 to n)
      for (j = 1 to n)
         for (k = 1 to n)
            a[i,j] = min(a[i,j], a[i,k] + a[k,j]);
   return a;
}
```

A. Provide an example of a graph where this implementation produces an incorrect result.

B.Determine which of the following statements is true:

1. The output of the above algorithm can be smaller than the correct result but cannot be larger.
2. The output of the above algorithm can be larger than the correct result but cannot be smaller.
3. The output of the above algorithm can be either smaller or larger than the correct result.

In the new algorithm, the update operation a[i,j] = min(a[i,j], a[i,k] + a[k,j]); legitimately corresponds to replacing a previous best path from i to j by a path that goes through k. Therefore, the new algorithm is indeed finding the cost of some path from i to j. The problem is that it is overlooking some paths and not finding the best path. Therefore (ii) is correct for B.



To answer (A) it is necessary to find a case where the new algorithm overlooks the best path. With a little thought, you can see that in the following graph:

the path 3→4→5 is not found until i = 3, j = 5; and at that point it will not be reconsidered in the calculation of the path from 1 to 2.

Q1.The 4SUM problem takes as input a set of real numbers and asks if there are four numbers whose sum is 0. The 3SUM problem similarly asks if there are three numbers whose sum is 0. Consider this algorithm to reduce from 3SUM to 4SUM.

def reduce3SUMto4SUM (set S):
return S U {0} // add 0 to the set

State the correctness lemma that this algorithm should satisfy for it to be a correct reduction. Then either prove or disprove this lemma.

[2.5 points] Lemma: {a1 a2 ... an} has 3 numbers whose sum is 0 iff {a1 a2 ... an 0} has 4 numbers whose sum is 0.

[2.5 points] The lemma is incorrect. Suppose S = {-1,-2,1,2}. Sum of all elements of S is 0. Hence, the sum of any three is never 0. So S does not have 3 numbers that add to 0. However, {-1,-2,1,2,0} has 4 numbers {-1,-2,1,2} that add to zero. This disproves the lemma.