1 Q1

Following the next four steps for verification and if doesn't meet then we could know this algorithm is not optimal.

1)First for the basic comparasion, check the source(starting code)'s d value and predecessor. It should be s.d = 0 and s.π = NIL(as we know source should not have the predecessor)

2)Then we can traverse all the other vertex to verify whether the weight and predecessor is right, that's to say that they should meet the following equations:

$$v.d = v.\pi + w(v.\pi, v)$$

3) After the above step we can verify the connecting point's relation, as they have predecessor, but some points which may not have the similar connecting. If they don't have predecessor, then the value $v.\pi$ should be NIL and the value $v.d$ should be ∞

4) For any verification of the above step, if it doesn't mean the requirement, then we know it is not the shortest path's solution since they're the basic property of shortest path, the complexity of these steps are $O(V)$. After that, we need to verify that it's optimal, which is shortest, we need to apply each pass of Bellman-Ford, which is just like each pass of the graph, and its complexity is $O(E)$, if it changes then we know it's not optimal and could be reduced again. So the total complexity is $O(V + E)$.

2 Q2

In Dijkstra's algorithm, we start from a fixed source whose weight is zero and it has no predecessor, all other nodes are initialized as the ∞.

However, here the difference is that we should start from a given source, not the source point above. Since all other nodes are ∞, we cannot apply the original algorithm to do the relaxation. From the question, we know that the weight is W at most, so the maximum value of the longest path is $(V-1)W$, which is smaller than ∞

So we could use the priority queue with the size of $(V-1)W$ buckets, any vertice's distance with this given source s could be no more than $(V-1)W$, so they can all be found in these number of buckets. So the algorithm should be:

1) initialize all vertice to be ∞ 2) scan the buckets from 0 to $(V-1)W$, when finding a non-empty bucket, remove the vertex and relax the adjacent points

Keep doing the step 2) until it reaches the end, so the complexity is $O(VW)$, and the relax times for all the edges is also counted for size $O(E)$, so the total is $O(VW + E)$.

```
Dijkstra(G, w, s)
        INITIALIZE–SINGLE–SOURCE(G, s)
        S <— { }
        Q <— V[G]
```

```
while Q != {}
        u <- EXTRACT-MIN(Q)
        S <- S U {u}
For each vertex in $v_{Adj}[u]$
        Do RELAX(u, v, w)
```

3 Q3

The solution is as follows:

a) For every edge in the G_f, as we know from the definition of G_f and G_b, we could know the edge set E_f and E_b are the special edges with points assorted. In G_f, each edge's start point is less than the ending point, so there is no possibility that there exist an circle which contain the edge that could make the endpoint smaller than the start point. In the same way, G_b is also acyclic. For another aspects, since the i and j are both the vertex index, starting from v_1 to $v_{|V|}$, all the points should be from small to large, so it is like v_1, v_2 ... $v_{|V|}$, so this meets the topological sort order of $< v_1, v_2, ...v_{|V|} >$. In the same way, we could get similar conclusion for G_b.

b)According to the questioin, when iterating through all of the edges in a phase of Bellman-Ford, first iterate through the edges in G_f in ascending order of vertex number, then iterate through the edges in G_b in descending order of vertex number. They're all in that kind of topological sort.

When iterating through the edges in G_f, any path in the shortest-path-tree that starts at a vertex with the correct d[] value and uses only edges in A results in the correct d[] value; The number of passes needed is the maximum number of $G_f - G_b$ alternations on a path, which is at most (V+1)/2. Thus, the number of passes is at most (V+1)/2 instead of V, which is almost the same as $\lceil |V|/2 \rceil$.

c) Yes, according to the above prove in b), we could know that it improves the running time and reduce the computation by half.

4 Q4

First we can determine the presents of a negative-weight cycle in a graph, the method is by checking the diagonal of the matrix L^{n-1} in the ALL-PAIR-SHORTEST-PATH. If this diagonal contains any negative number, there must a negative-weight cycle in this graph. Then after detecting the circle, we need to find the length. If L^m is the first time that it occurred when we see the negative value, then its length for this cycle is m.

By using the SLOW-ALL-PAIRS-SHORTEST-PATH, we could directly get the length m. (we could also use a binary search in FASTER-ALL-PAIRS-SHORTEST-PATH)

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
        n = W.rows
```

```
        L1 = W
        for m = 2 to n - 1
                let Lm be a new nxn matrix
                Lm = EXTEND-SHORTEST-PATHS(L(m-1), W)

                if(Lm < 0)
                        return m

return L(n-1)

EXTEND-SHORTEST-PATH(L, W)
        n=L.rows
        L'=(lij') be a new nxn matrix
        for i=1 to n
                for j=1 to n
                        lij'=INFNITE
                                for k=1 to n
                                        lij'=min(lij', lik + wkj)

return L'
```

5 Q5

For a simple way, we could just run the FLOYD-WARSHALL algorithm one more time to see it the value changed, since if there exist a negative-weight circle it should decrease for the d values. Otherwise it will remain unchanged.

On the other solution, which is more precise and efficient is that, check the main diagonal value for the result matrix, similar to the above solution, if there is a negative value and meets the following requirements, then it has:

1) check d_{ii}^n to see whether it is smaller than zero then there is a negative path from i to iteself.

2) for other cases like more than or equal to one vertex, it still applies for the judgement in 1). Let's assume that there exist at least two vertices in the shortest path, and k is the largest number vertex in this cycle, then d_{ik}^{k-1} and d_{ki}^{k-1} must have corrected the shortest path since they don't count on the negative-weight value. So i -¿ k -¿ i is a negative circle, the sum of those two weights d_{ik}^{k-1} and d_{ki}^{k-1} must be negative, but as we see that their value is non-negative since the d value can never decrease, that's to say that between point to other points must be non-negative if doesn't contain i to itself path. So the only way is that the i to itself path is negative, which is the same as the 1) said.

In these proof, we can get to know that the assumption is correct and we still just need to check the diagonal value of the matrix for d_{ii}^n and if there is a negtive value, then it has a negative-weight circle.