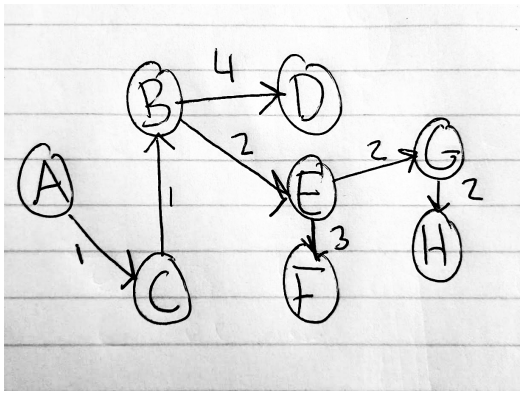
Problem 0.
https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/
https://www.geeksforgeeks.org/dsa/why-does-dijkstras-algorithm-fail-on-negative-weights/

Problem 1.
a. Distance from source to each node =

Step	**Min heap (dist, node)**	A	B	C	D	E	F	G	H
0	(0, A)	0	∞	∞	∞	∞	∞	∞	∞
1	(1, C), (3, B)	0	3, A	**1, A**	∞	∞	∞	∞	∞
2	(2, B), (8, F)	0	**2, C**	1, A	∞	∞	8, C	∞	∞
3	(4, E), (6, D), (8, F)	0	2, C	1, A	6, B	**4, B**	8, C	∞	∞
4	(6, D), (6, G), (7, F)	0	2, C	1, A	**6, B**	4, B	7, E	6, E	∞
5	(6, G), (7, F)	0	2, C	1, A	6, B	4, B	7, E	**6, E**	∞
6	(7, F), (8, H)	0	2, C	1, A	6, B	4, B	**7, E**	6, E	8, G
7	(8, H)	0	2, C	1, A	6, B	4, B	7, E	6, E	**8, G**

Order at which permanently labeled: A, C, B, E, D, G, F, H
Image of shortest path tree:



b.
Because of the negative edge weight between E and C, Dijkstra's algorithm will not give the correct shortest path tree.

Consider the shortest path given by Dijkstra's to C is 1. This path is from A to C. However, if we were to go from A to C (1) to B (1 + 1), to E (1 + 1 + 2), back to C (4 - 4), we would get a path length of 0, which is less.

c. Bellman-ford algorithm
Examine outgoing edges at each step

iteration	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	~~3~~ 2	1	7	5	8	~~12~~ 7	9
2	0	2	~~1~~ 0	6	4	7	6	8
3	0	1	0	6	4	7	6	8
4	0	1	-1	5	3	7	5	7
5	0	0	-1	5	3	6	5	7
6	0	0	-2	4	2	5	4	6
7	0	-1	-2	4	2	5	4	6

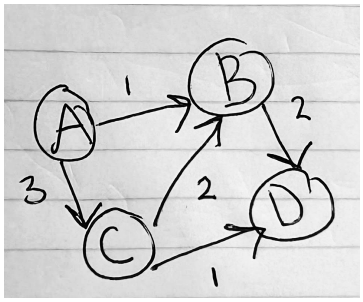
There is a negative cycle because after |v| - 1 iterations, there is another edge where d[u] + w < d[v].

Problem 2.
a.
This algorithm finds the shortest path distance from s to all nodes.
For each node u (in topological order ie, parent before child), it goes through every child v, and if the distance from s to u + weight of edge (u,v) is less than the current distance s to v, it sets the distance from s to v to be s to u + weight of edge (u,v) (ie, it relaxes it).

Because it is in topological order, children will only be visited after their parents, therefore, when it's checked if (s,u) + (u,v) < (s,v), (s,u) will already be the shortest possible distance because it would've already been relaxed if it could have been.

b.
O(|E|) because you check each edge only once. Once you check the outgoing edges of a node, you move on to the children, and won't check that edge again.

c.
In the graph with vertices V = A, B, C, D and edges E = (A, B, 1), (A, C, 3), (C, B, 2), (C, D, 1), (B, D, 2)



Both algorithms would first inspect A, and find the weights (B, 1) and (C, 3). However, Dijkstra's would explore B next because it is implemented with a minimum heap, while the prob 2 algorithm would explore C next because of the topological ordering.

Although the steps are different, they will still produce the same shortest distances

Dijkstra's:

it	**Min heap**	**A**	**B**	**C**	**D**
0	(A, 0)	0	∞	∞	∞
1	(B, 1), (C, 3)	0	1	3	∞
2	(C, 3), (D, 3)	0	1	3	3
3	(D, 3)	0	1	3	3
4	-	0	1	3	3

Problem 2 Algorithm:
Topological sort A C B D

Node being inspected (u)	**A**	**B**	**C**	**D**
-	0	∞	∞	∞
A	0	1	3	∞
C	0	1	3	4
B	0	1	3	3
D	0	1	3	3

Problem 3.

Create a new graph G_1.
Add every node from G to G_1.

For every edge (u, v, w) in G

add an edge (u, v, 0) to G_1 //ignoring original edge weight

add an edge (v, u, 1) to G_1 //reverse edge with weight of 1

Then run dijkstra's to find the min path to vertex n on graph G_1.
The weight it returns will be the number of flips.