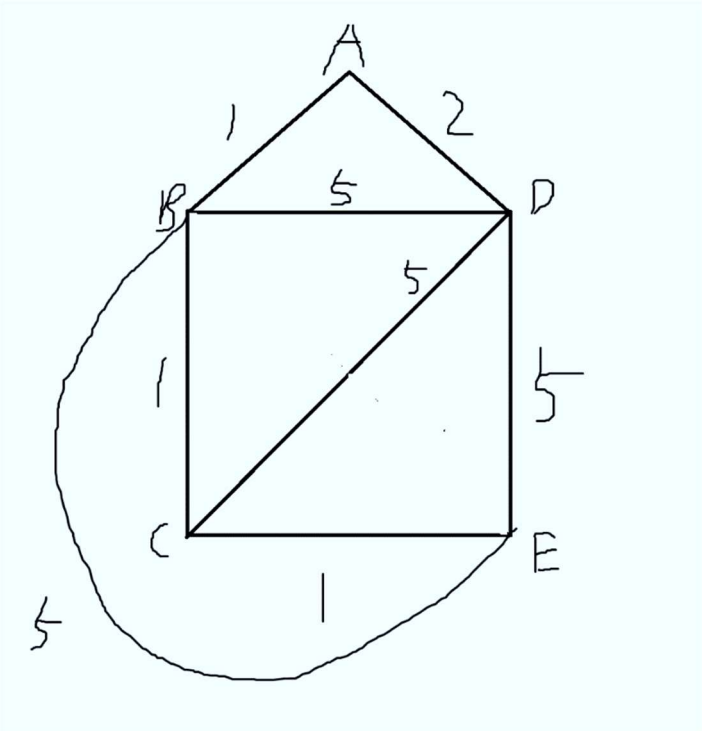
Q1:

a)

In the graph G below:



Set the starting vertex to be A.

The minimum spanning tree is $\{AB, AD, BC, CE\}$ with weight $1 + 1 + 1 + 2 = 5$.

In BFS, edge CE can never be included. This is because all vertices are at most 2 edges away from A, and BFS is completed when it has visited all vertices. That means it will not try to include an edge that is between two vertices which both are height 2.

In DFS, the spanning tree will only include one of $\{AB, AD\}$. This is subgraph BCDE is a clique, and DFS will return a spanning tree that looks like a list (can be proved by induction since a clique after removing one vertex is still a clique). After traversing subgraph $BCDE$ the spanning tree is done, and only one of $\{AB, AD\}$ is visited.

Therefore, neither the DFS or BFS will give a minimum weight spanning tree of G.

b)

we prove the contrapositive of the statement.

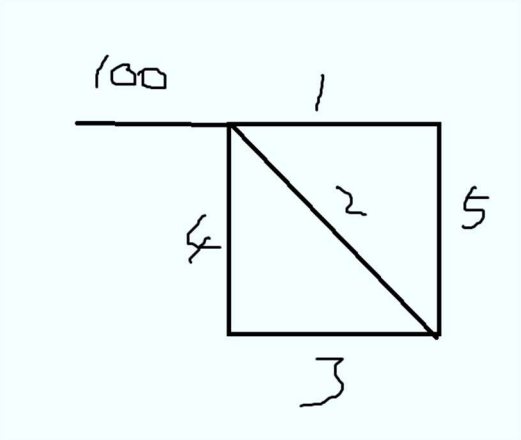
If G is not a tree, the G contains at least one cycle. Since in DFS all vertices in this cycle will be descendants of one vertex in this cycle, in BFS at least two vertices in this cycle have a same parent

vertex(either in the cycle or outside the cycle), they will produce a different spanning tree for G. Hence the contrapositive is true, the original statement is true.

Q2:

a)

Statement is false.



In the graph above there are $n = 5$ vertices and $m = 6$ edges. The heaviest edge must be included since it is a bridge.

b)

Statement is true.

By the first step of Kruskal's algorithm, the lightest edge will be chosen.

c)

Statement is true.

We can find a minimum spanning tree H for G without vertices from the cycle, and a minimum edge f that connects the cycle and H(f exists since graph G is connected). The minimum spanning tree T for the cycle is all edges of the cycle except e. Then the union of H, f, T is a minimum spanning tree for graph G, and it does not contain edge e.

d)

Statement is true.

This is because there are only comparisons being done in Prim's algorithm, and negative weight works fine in comparisons. Only operations such as addition, subtraction that compute using weight will possibly not work with negative weights.

Q3:

We will use modified version of Bellman-Ford Algorithm. Instead of adding just weight of the edge, we also add the weight of the vertex.

```
BellmanFord_Modified(G = (V; E); w; s) {
        For all v ∈ V do {
                If (s, v) ∈ E then {
                        d[v] = c(s) + w(s, v) + c(v);
                } Else if v == s then {
                        d[v] = c(s);
                } Else {
                        d[v] = +∞;
                }
        }
        For i = 2,3, … , n − 1 do {
                For all (u, v) ∈ E do {
                        if d[u] + w(u, v) + c(v) < d[v] then {
                                d[v] = d[u] + w(u, v) + c(v);
                        }
                }
        }
}
```

The correctness follows directly from Bellman-Ford Algorithm discussed in class. We initialized the original array adding the cost of the vertex it has past, and update the array according to the weight of both the edge and the vertex, so that the path contains the cost of every vertex in it.

Since only the value assignment is changed, the algorithm still has running time $\theta(mn)$.

Q4:

i)

let vertices be triplet (a, b, c) where $\begin{cases} a+b+c=11 \\ 0 \le a \le 4 \\ 0 \le b \le 7 \\ 0 \le c \le 10 \end{cases}$, directed edge between vertices exists if one can perform exactly one "pour" operation and arrive at the other vertex. Example is that $(4,7,0)$ can go to $(0,7,4)$ by pouring the first jar into the third jar. Note this is directed because some operations are irreversible, for example $(2,2,7)$ can point to $(0,4,7)$ by pouring from the first to the second, but $(0,4,7)$ cannot point to $(2,2,7)$ since pouring cannot stop until one is empty or full. If there exists a path from $(4,7,0)$ to $(2, b, c)$ where $\begin{cases} b+c=9 \\ 0 \le b \le 7 \\ 0 \le c \le 10 \end{cases}$, then there exists a sequence of directed edges(operations) that can leave the first jugs $2L$ of water.

ii)

The Bread-First-Search algorithm will find the shortest path, indicating the least number of operations.

iii)

We will use Dijkstra's algorithm, assigning each edge the weight of the number of liters of water poured. For example, $(4,7,0)$ to $(0,7,4)$ has weight 4, $(2,2,7)$ to $(0,4,7)$ has weight 2. Then the distance of the path from $(4,7,0)$ to $(2, b, c)$ where $\begin{cases} b+c=9 \\ 0 \le b \le 7 \\ 0 \le c \le 10 \end{cases}$ will be the total number of liters of water that need to be transferred. That number divided by $1L/min$ gives the time that is required to finish the pouring.

Q5:

$\text{Dynamic Relay Race}(n)$ calculated the minimum time that is need to complete a Dynamic Relay Race with n check points. We use a helper function $DRR(n, a, b)$, and let $\text{Dynamic Relay Race}(n) = DDR\left(n, \frac{n}{2}, \frac{n}{2}\right)$. In DDR, a and b sums to n, where a indicated the number of segment that Alice is allowed to run, b indicates the same for Bob.

The subproblem we want to solve is how much time is required for $n-1$ segments.

First two cases are Alice run the last segment, last two cases are Bob run the last segment,

$$DRR(n, a, b) = minimum \begin{cases} DRR(n-1, a-1, b) + a_n & if\ last\ segment\ is\ run\ by\ Alice \\ DRR(n-1, a-1, b) + a_n + p_i & if\ last\ segment\ is\ run\ by\ Bob \\ DRR(n-1, a, b-1) + b_n & if\ last\ segment\ is\ run\ by\ Bob \\ DRR(n-1, a, b-1) + b_n + p_i & if\ last\ segment\ is\ run\ by\ Alice \end{cases}$$

To do recursion, we will need a global variable to indicate who run the last segment to apply the switching penalty rule.

If $a = 0$, the rest segments must be run by Bob, vice versa.

Running time:

We need to fill up an $n/2$ by $n/2$ grid, each cell takes $\theta(1)$ time, in total the running time is $\theta(n^2)$.