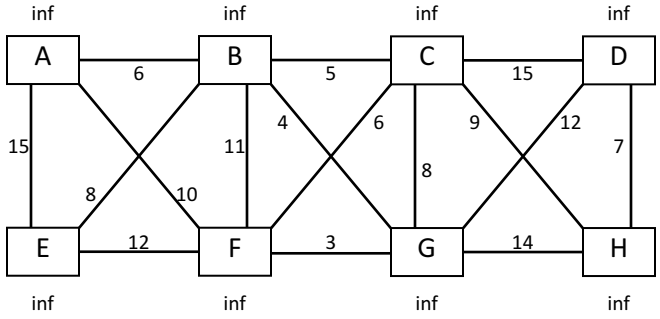
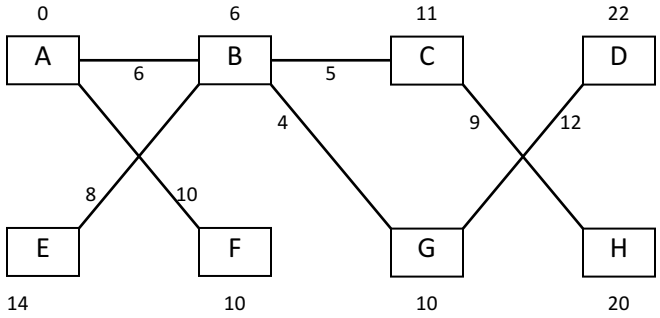
Name: Md. Habibur Rahman Rony
Student ID: 984582
Weekday: Week 3- Day 14

Answer to the Q. No.7-1:

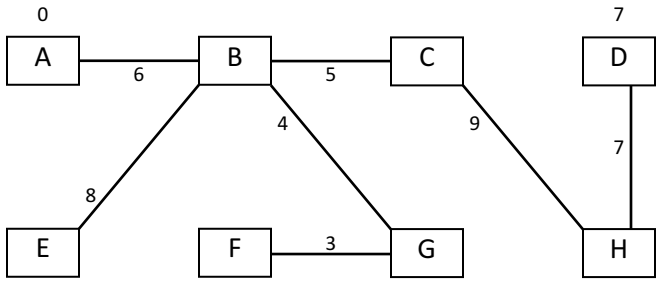


	A	B	C	D	E	F	G	H
A	0	**6_A**	Inf	Inf	15_A	10_A	Inf	Inf
B	0	6_A	**11_B**	Inf	**14_B**	10_A	10_B	Inf
F	0	6_A	11_B	Inf	14_B	10_A	10_B	Inf
G	0	6_A	11_B	22_G	14_B	10_A	10_B	24_G
C	0	6_C	11_B	22_G	14_B	10_A	10_B	**20_C**
E	0	6_C	11_B	22_G	14_B	10_A	10_B	20_C



Answer to the Q. No.7-7:

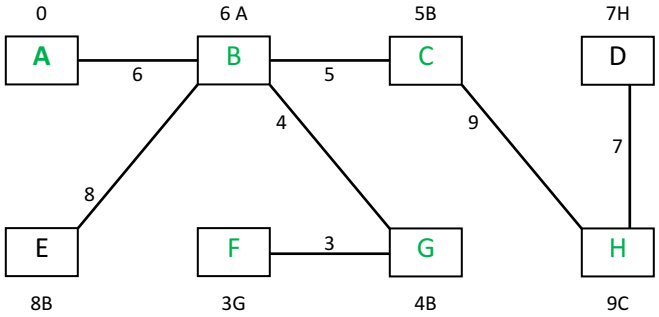
	A	B	C	D	E	F	G	H
A	0	6_A	Inf	Inf	15_A	10_A	Inf	Inf
B	0	6_A	**5_B**	Inf	**8_B**	10_A	4_B	Inf
G	0	6_A	5_B	12_G	8_B	**3_G**	4_B	14_G
F	0	6_A	5_B	12_G	8_B	3_G	**3_G**	14_G
C	0	**5_C**	5_B	12_G	8_B	3_G	3_G	**9_C**
E	0	5_C	5_B	12_G	8_B	3_G	3_G	9_C
H	0	5_C	5_B	**7_H**	8_B	3_G	3_G	9_C



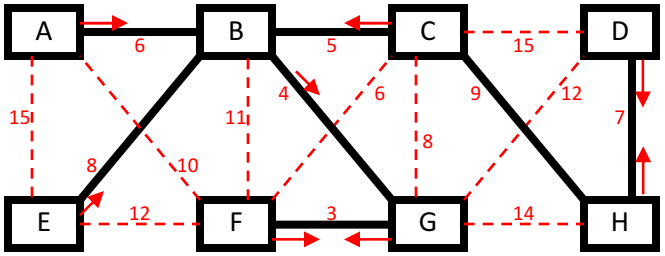
[{F,G}=3, {G,B}= 4,{B,C}=5,{B,A}=6,{D,H}=7, {B,E}=8, {C,H}=9]
So, total weight = 3+4+5+6+7+8+9=42

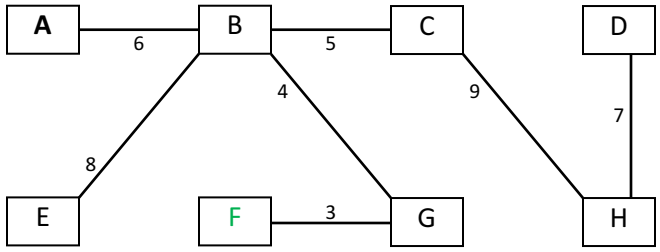
Answer to the Q. No.7-8:

A-B	6
B-G	4
G-F	3
C-H	9
H-D	7
E-B	8
D-G	12



Answer to the Q. No.7-9:





Answer to the Q. No.a:
This algorithms will not successfully computes MST, because we don't have any checking whether the edge that we remove will give result of more than 1 connecting component. If after we remove the edge, it will make additional

connecting component cycle will still happen even though the number of edges are n-1.

Answer to the Q. No.b:

This algorithms will not successfully computes MST, because since we take each edges in arbitrary order, we might take the edge with the maximum number, which makes it is not a minimum spanning tree.

Answer to the Q. No.c:

This algorithms will not successfully computes MST, because we don't have any checking whether the edge that we remove will give result of more than 1 connecting component. If after we remove the edge, it will make additional connecting component cycle will still happen even though the number of edges are n-1.

Answer to the Q. No.C-3-28

To implement the method before and closestBefore in a skip-list, we have to make sure that the skip list has a pointer to the element before. Then find l, and then call the pointer down, until we go to the bottom element, and call the pointer before the element in the current pointer.
And the expected running time will be O(log n) because we might need to traverse all the element to find l and after that to get the before l, we need constant time, so it won't be counted, which will results O(log n)

Answer to the Q. No.C-5-1

Guessing 1, the water holes are located along the road and we need to sort the closest refill water place.

```
Algorithm RefillWater (p, k)
        Input : p is water refilling place, k miles for one bottle
        Output : p2 is path with less stops for refill

        PQ <- new heap priority queue
        for i<--0 to p.size() do
                PQ.insert(p.elemAtRank(i), p.elemAtRank(i)))

        tem <-- 0
        while ! PQ.isEmpty()  do
                tem <-- tem + PQ.removeMin()
                if  tem + PQ.peekMin() < k then
                        p2.insertElement(tem)
                        continue
                p2.insertElement(tem)
                tem <-- 0
        return p2
```