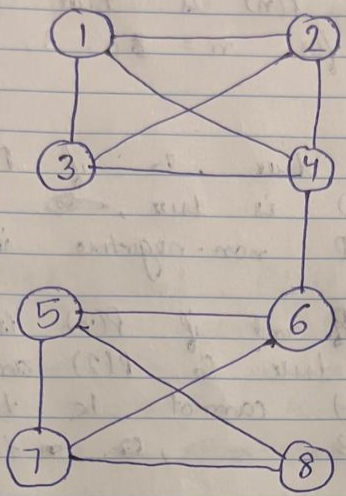
Exercise 1:

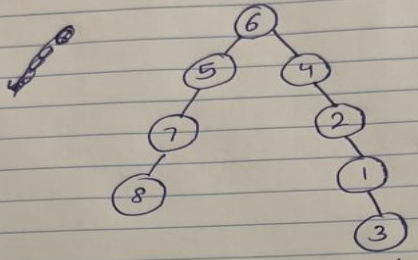
Exercise-1

1.



2. DFS ~~starting~~ at 6.

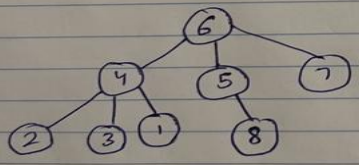
6, 5, 7, 8, 4, 2, 1, 3



So, in DFS, we go in depth of this and backtracking if its already visited.

3. BFS starting at vertex 6.

6, 4, 5, 7, 2, 3, 1

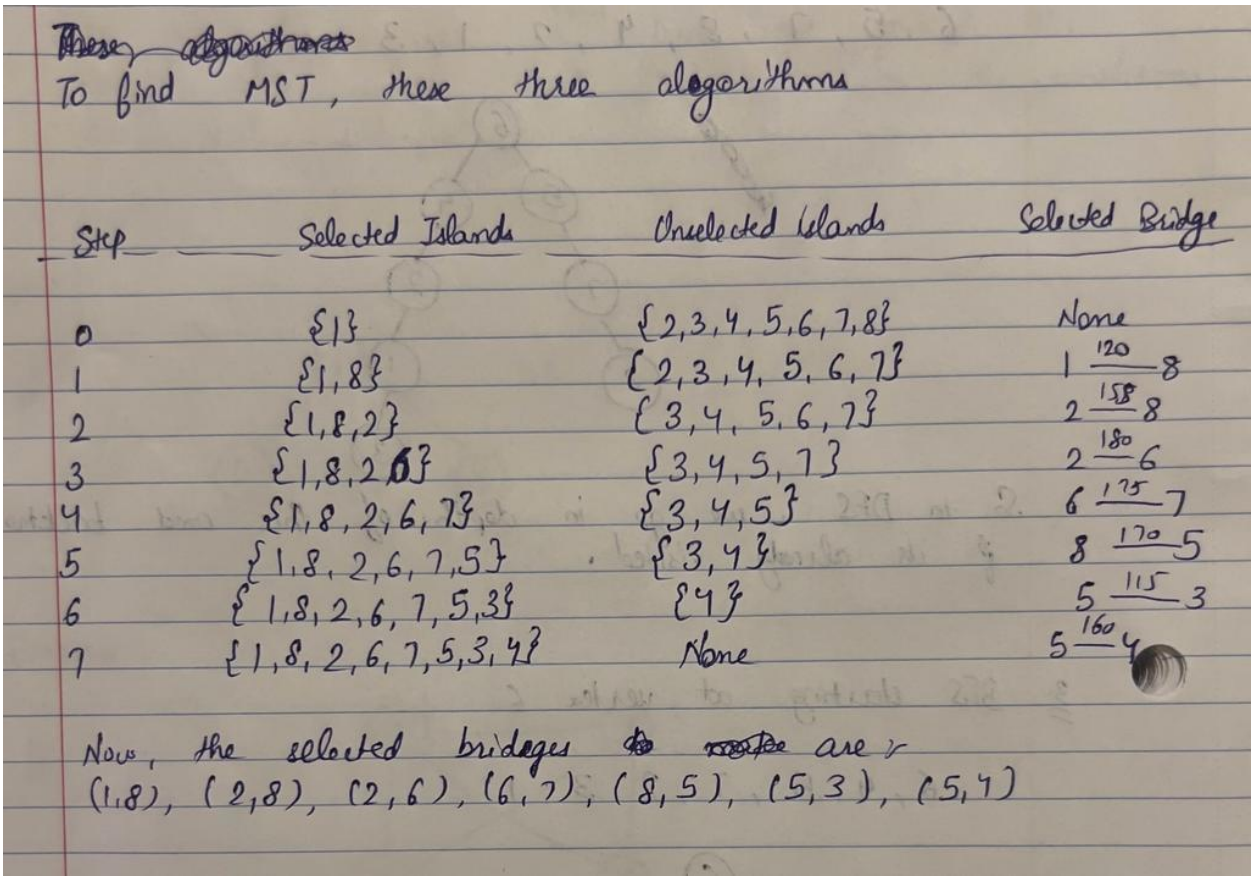


In BFS, first, we explore ~~one~~ one vertex and add it to queue and then we check next item in queue and explore their adjacent ~~item~~ vertexes if its not visited.

Exercise 2:

To find the shortest path, these three algorithms are used if the weight is positive but for a non-positive weight, Kruskal and Prim's algorithm gives the shortest path with positive and negative labels because both algorithms select the edges with minimum weight. For Dijkstra algorithm, it cannot detect the negative weights cycles, which gives the incorrect results at the end, so shortest path cannot be formed with this algorithm.

Exercise 3:



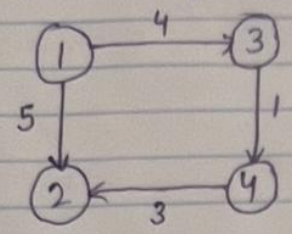
To find MST, these three algorithms

Step	Selected Islands	Unselected Islands	Selected Bridge
0	{1}	{2,3,4,5,6,7,8}	None
1	{1,8}	{2,3,4,5,6,7}	$1 \xrightarrow{120} 8$
2	{1,8,2}	{3,4,5,6,7}	$2 \xrightarrow{158} 8$
3	{1,8,2,6}	{3,4,5,7}	$2 \xrightarrow{180} 6$
4	{1,8,2,6,7}	{3,4,5}	$6 \xrightarrow{175} 7$
5	{1,8,2,6,7,5}	{3,4}	$8 \xrightarrow{170} 5$
6	{1,8,2,6,7,5,3}	{4}	$5 \xrightarrow{115} 3$
7	{1,8,2,6,7,5,3,4}	None	$5 \xrightarrow{160} 4$

Now, the selected bridges are :
(1,8), (2,8), (2,6), (6,7), (8,5), (5,3), (5,4)

Exercise 5:

No, the greedy algorithm does not always find the shortest path from start to goal. For example :-



Here, if we find a ~~shortest~~ path from 1 to 2 using the given following strategy, we have 2 edges,
$1 \rightarrow 3$ with weight 4 and
$1 \rightarrow 2$ with weight 5.

According to algorithm, it goes with minimum weight which is 4. So, path is $1 \rightarrow 3$ and then $3 \rightarrow 4$ with weight 1 and $4 \rightarrow 2$ with weight 3. So, total weight according to algorithm is $4 + 1 + 3 = 8$

which is not a shortest path because shortest path with less weight is $1 \rightarrow 2$, weight = 5.

Exercise 6:

1. In Boolean matrix, to deleteEdge(i, j), it takes a constant time O(1) to the edge and to delete it.
   In doubly-linked list to deleteEdge(I, j), first we have to find the edge to delete which takes n time complexity and constant time to delete it. So, total time complexity is O(n).

2. In Boolean matrix to deleteEdge(i, j), it takes n time complexity to find vertex i and to delete their nodes. So, total time complexity is O(n).

For doubly linked list, to find the node to delete it takes n time complexity and to delete their connected nodes it again take n time and to delete, it takes constant time. So, it takes $O(n^2)$ time complexity.