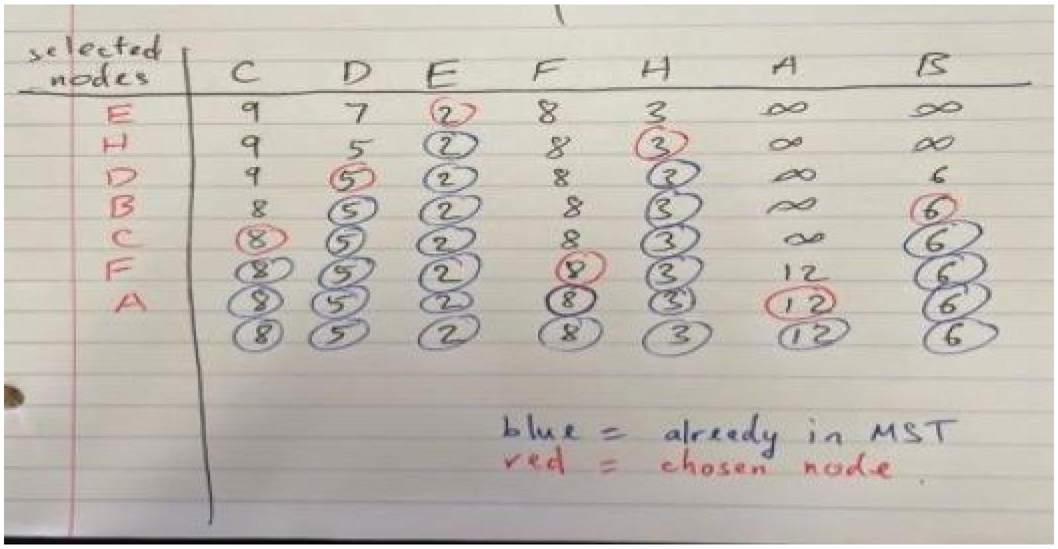
Problem 1:

a) The Dijkstra algorithm would be the best algorithm for finding the shortest path route from the distribution center to each of the towns. From G, we first pick the shortest path vertex that is neighboring to G which, in this case, is E since the distance from G to R is 2. Then we update the shortest path to other vertices. We repeat this process until all vertices are relaxed. Included below is the result table.



selected nodes	C	D	E	F	H	A	B
E	9	7	2	8	3	∞	∞
H	9	5	2	8	3	∞	∞
D	9	5	2	8	3	∞	6
B	8	5	2	8	3	∞	6
C	8	5	2	8	3	∞	6
F	8	5	2	8	3	12	6
A	8	5	2	8	3	12	6
	8	5	2	8	3	12	6

blue = already in MST
red = chosen node

Vertex	Distance	Shortest path from G
C	8	G->E->D->C
D	5	G->E->D
E	2	G->E
F	8	G->F
H	3	G->H
A	12	G->E->D->C->A
B	6	G->H->B

b) Pseudocode:

```
Dijakstra(G ,w,s)
        InitializeSingleSource(G, s)          // O(V)
        S ←∅
        Q ← V[G]
        While Q ≠ 0 do                      // this loop executes O(V) times
            u ← ExtractMin(Q)
            S ← S∪{u}
            For u ∈ Adj[u] do               // this loop executes O(E) times
                Relax(u,v,W)

    InitializeSingleSource(G, s)            // this function takes O(V) times
        For v ∈ V[G] do
            d[V] ← ∞
            p[v] ← 0
        d[s] ← 0
    Relax(u, v, w)                      // this function executes O(1)
        If d[v] > d[u] + w[u,v] then
            d[v] ← d[u] + w[u,v]
            p[v] ← u
```

So the time complexity of this algorithm is O(E*V).

c) In the graph above E would be the optimal town to locate the distribution center because the farthest route from E (E->A) gives a distance of 10 which is optimal among other distribution centers.

d) I would pick a random vertex to be the first distribution town, and then choose the farthest path to another location from the first distribution town to be the second distribution center so that I could eliminate the distance from the first distribution town to the second one.

e) In the graph above, the two optimal town to place the distribution centers would be G and A.

Problem 2:

For this problem, I'm going to use Prim's algorithm that takes a 2D matrix as an input to find the MST since the number of vertices isn't too big. In the algorithm, I will have 3 arrays, isIncluded[] is of type boolean to check which vertices are already included in the MST, key[] is of type int which stores values of all vertices, and p[] is of type int where each element is the parent of the associated index. I will first initialize all the elements of key[] to infinity and all the elements of isIncluded[] to false. Then I will have a loop to loop n (number of vertices) times. Each time, I will find the vertex with the smallest key value that is not included in the MST, then I will update the key value of the current vertex to the edge of that vertex to its parent if it is greater than 0 and smaller than infinity. This vertex is now set to true in isIncluded[] since it is added to the MST. this process is repeated until all the elements are set to true in isIncluded.

Pseudocode:

```
MST-Prim(G)
    for each i in key
        key[i] = infinity
    for each i in isIncluded
        isIncluded[i] = false
    key[0] = 0                    // set first element to 0 -> starting vertex
    Create p array
    p[0] = null    //    parent of starting vertex is null
    While (i = 0 < number of vertices)
        u = ExtractMin(key, isInclude)    // get the vertex with smallest key
        isIncluded[u] = true         // u is included in MST
        for v from 0 to number of vertices - 1
            if (u is not in MST)
                If (G[u][v] >0 and G[u][v] < infinity)
                    p[v] = u
                    key[v] = G[u][v]

        i++
    for j is 0 to number of vertices -1
        sum = sum + G[j][p[j]]
    return sum
```

Theoretical running time of this algorithm would be O(V^2) because we have 2 nested loops that each runs V times.