Problem #1: Optimum meeting place

```
def optimumMeetingPlace(G):
    dist1=dijkstra(G,1)
    dist2=dijkstra(G,50)
    minTotalDistance=math.inf
    meetingPoint=0

    for vertex in range(1,51):
        totalDistance=dist1[vertex]+dist2[vertex]
        if totalDistance<minTotalDistance:
            minTotalDistance=totalDistance
            meetingPoint=vertex

    return meetingPoint
```

Here in the optimumMeetingPlace(G) function is taking the graph as input and calling the given dijkstra(G,a) algorithm twice having 2 sources 1(your house) and 50(Benji's house). So these call-ins will return 2 arrays/lists containing the minimum distances to all the vertices from sources 1 and 50 .
Then using a loop , the minimum distance for a meet point(vertex) was calculated by summing the distances gained from returned array values . After that , which vertex's summing minimum distance will be the lowest becomes the optimum meeting point from both sources and returns the vertex.

Problem #2: Meet in the middle!

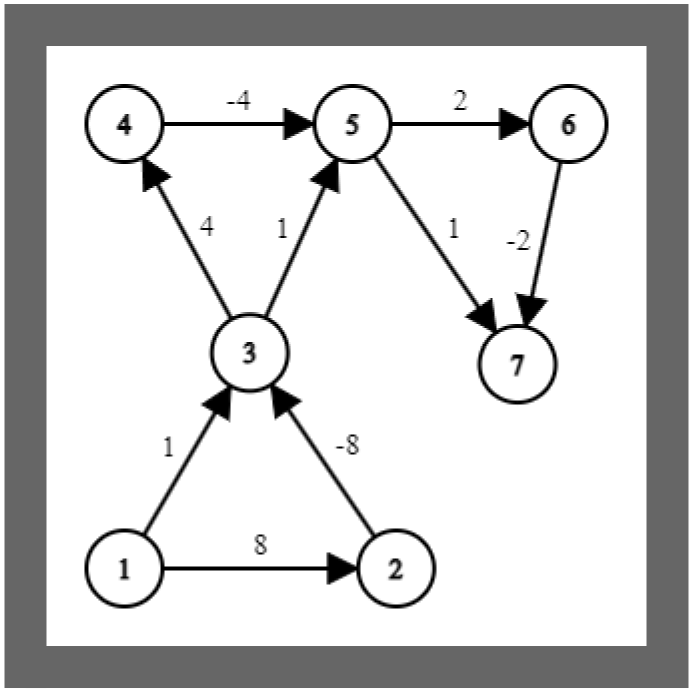
```
function optimumKFC(G,KFCareasList):
    dist1=dijkstra(G,1)
    dist2=dijkstra(G,50)
    minTotalDistance=math.inf
    optimumKFCarea=-1

    for vertex in KFCareasList:
        totalDistance=dist1[vertex]+dist2[vertex]
        if totalDistance<minTotalDistance:
            minTotalDistance=totalDistance
            optimumKFCarea=vertex

    return optimumKFCarea
```

Here the optimumKFC(G,KFCareasList) function is taking the graph and the list of areas(vertices) where KFC outlets are available. Now the function is performing the similar tasks as problem#1 except counting minimum summing distances from all vertices , counting minimum summing distances from KFC available vertices was done instead .

Problem #3: Dijkstra’s algorithm in a ‘negative’ weight graph!!



a. Applying dijkstra on the given Graph: (Queued)

Visited	1	2	3	4	5	6	7
1(0)	0	∞	∞	∞	∞	∞	∞
3(1)		8	1	∞	∞	∞	∞
5(2)		8		5	2	∞	∞
7(3)		8		5		4	3
6(4)		8		5		4	
4(5)		8		5			
2(8)		8					

b. As the dijkstra algorithm doesn't take the visited vertices in count while finding the minimum cost , some paths with negative cost ; which might lower the total minimum cost, will remain ignored as the visited vertices are dequeued from the queue .

Such a vertex is 7. Applying the dijkstra algorithm , we found that the cost reaching to 7 from source 1 is 3(1→3→5→7) . But we can see from the graph , there is another path with lower cost which is 2(1→3→5→6→7) is being ignored by the dijkstra algorithm .

c. If we apply the modified dijkstra algorithm on the graph: (Queued)

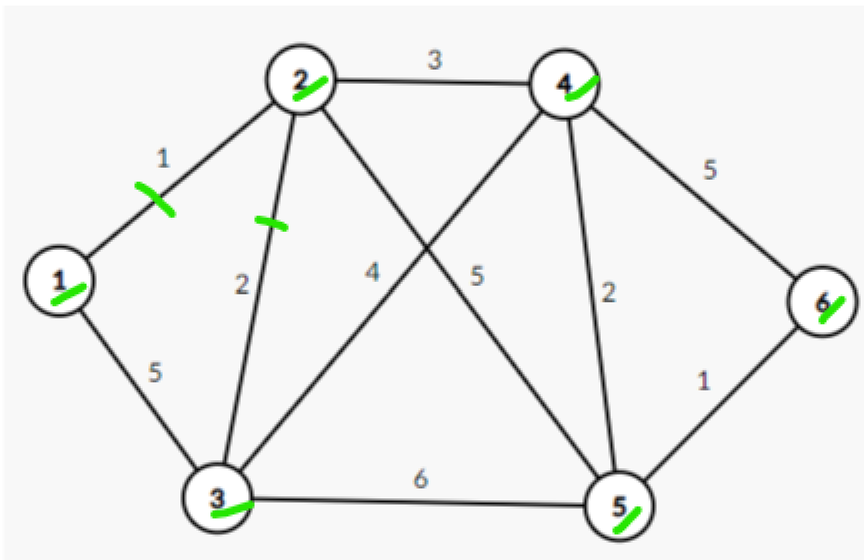
Visited	1	2	3	4	5	6	7
1(0)	0	∞	∞	∞	∞	∞	∞
2(8)	0	8	∞	∞	∞	∞	∞
3(0)	0	8	0	∞	∞	∞	∞
4(4),5(1)	0	8	0	4	1	∞	∞
5(0)	0	8	0	4	0	∞	∞
6(2),7(1)	0	8	0	4	0	2	1
7(0)	0	8	0	4	0	2	0

Yes , this time the modified dijkstra algorithm is taking in count the negative weight edges and finding the correct shortest path(1→2→3→4→5→6→7).

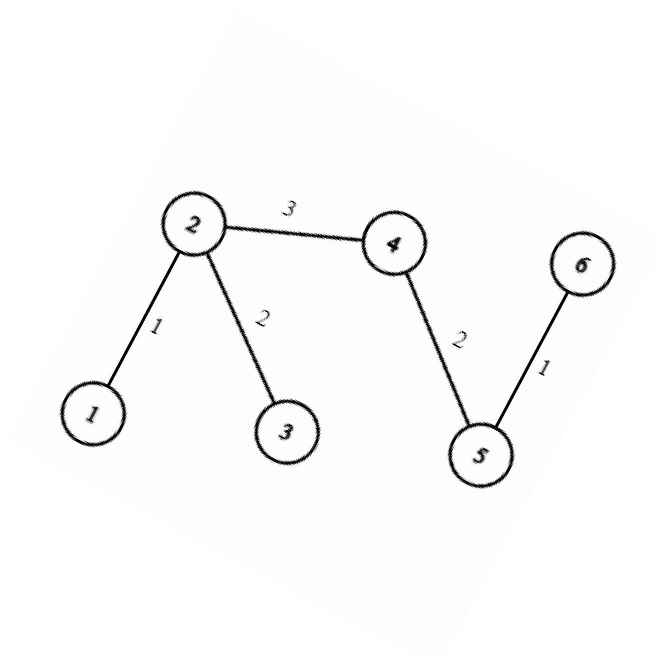
d. The time complexity for the updated dijkstra algorithm will be $O(V * E * log_2(V))$; as now we have to take in consideration all the vertices .

Problem #4: A divide-and-conquer MST algorithm?!

Applying the known **Prim's algorithm** on the example graph :

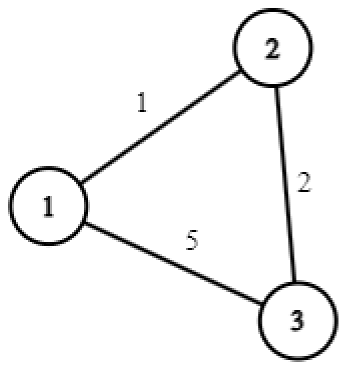


Priority Queue of edges ={(1,2,1),(2,3,2),(2,4,3),(4,5,2),(5,6,1),(1,3,5),(2,5,5),(3,4,4),(3,5,6),(4,6,5)}
MST:

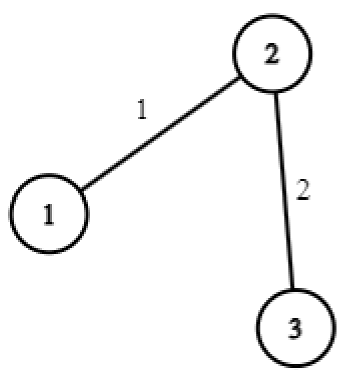


Output={(1,2,1),(2,3,2),(2,4,3),(4,5,2),(5,6,1)} ; Cost=9

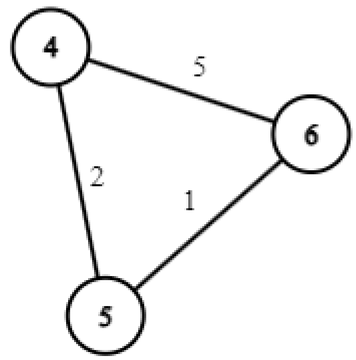
Now applying the **divide-and-conquer MST algorithm** :



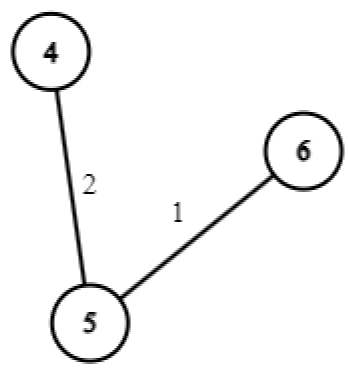
Priority Queue of edges of leftG ={(1,2,1),(1,3,5),(2,3,2)}



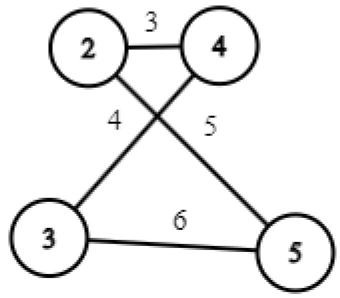
Output of the leftG={(1,2,1),(2,3,2)}



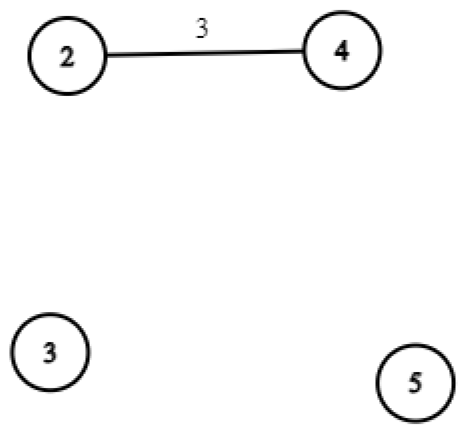
Priority Queue of edges of rightG ={(4,5,2),(4,6,5),(5,6,1)}



Output of the leftG={(4,5,2),(5,6,1)}

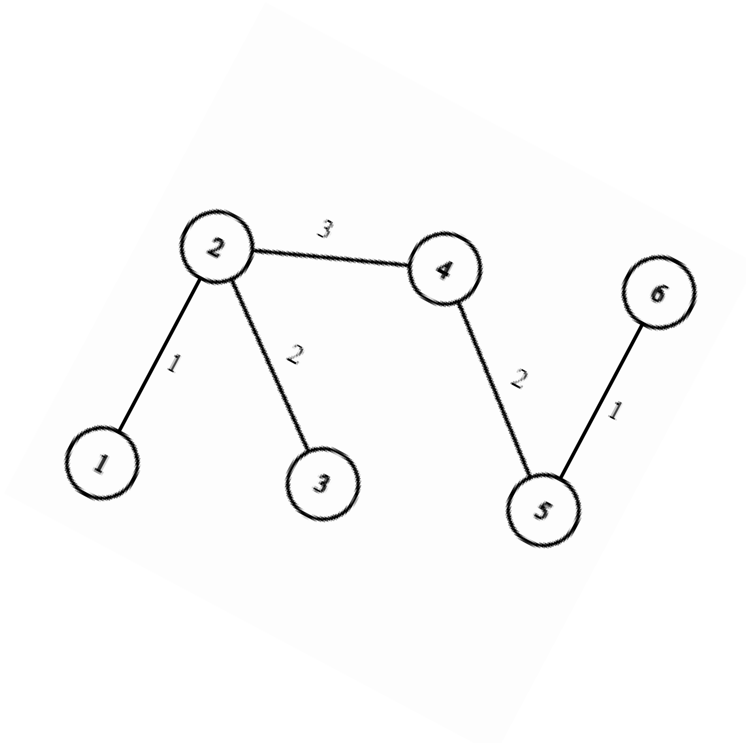


Priority Queue of edges of remainsG ={(2,4,3),(3,4,4),(2,5,5),(3,5,6)}



Output of the leftG={(2,4,3)}

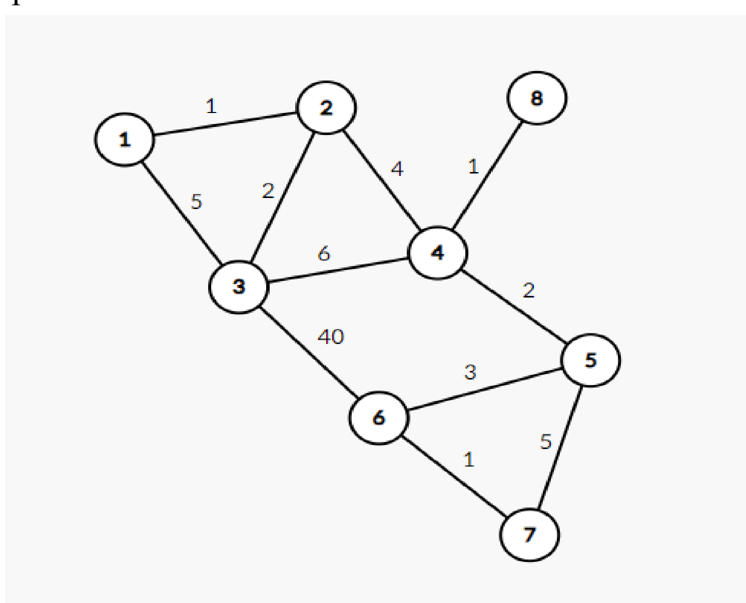
Total MST :



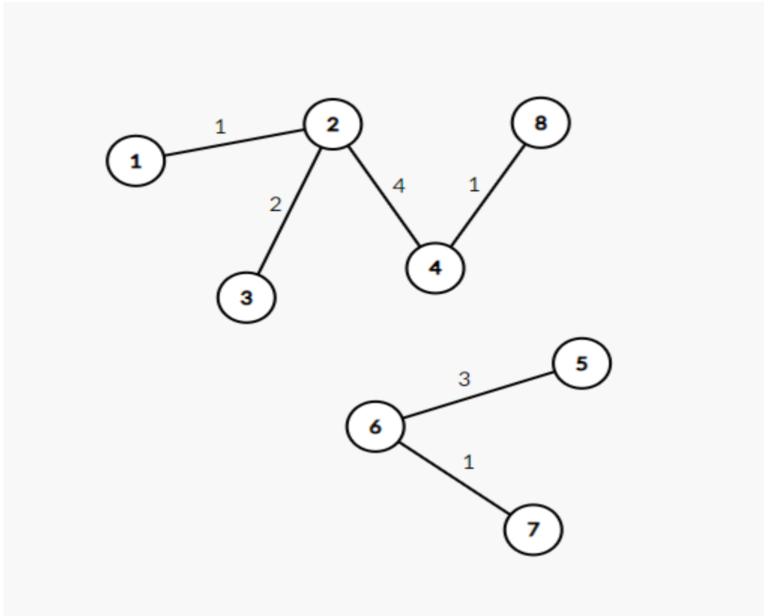
Output={(1,2,1),(2,3,2),(2,4,3),(4,5,2),(5,6,1)} ; Cost=9

For this example the algorithm provided right answer , but the algorithm won't show the right answer always:

Suppose given graph :



MST :



In this case, if we apply the algorithm:
leftGV={1,2,3,4}, rightG={5,6,7,8}, remainsG={(4,8,1), (4,5,2), (3,6,40)}

MSTleft={(1,2,1),(2,3,2),(2,4,4)}, MSTright={(5,6,3),(6,7,1)}
Choosing the edge (4,8,1) from remainsG to connect MSTleft and MSTright
The resulting MST is a disconnected tree , which violates the main characteristics of MST .

So the divide-and-conquer MST algorithm is not always correct .