kruskal's algo

```python
from collections import deque

def bfsWithWeights(graph, start):
    # Initialize the queue with the start node, with a weight of 0 and no parent (-1)
    queue = deque([(0, start, -1)])
    visited = set()  # Set to keep track of visited edges.
    bfs_order = []  # List to store the order of BFS traversal.

    # Perform BFS
    while queue:
        wt, node, parent = queue.popleft()  # Dequeue a node from the front of the queue.
        if node not in visited:  # If the node has not been visited
            if parent != -1:
                bfs_order.append((wt, node, parent))  # Append the node to the BFS order list.
            # Enqueue all adjacent nodes that haven't been visited.
            for neighbor in graph[node]:
                if (neighbor[0], node) not in visited and (node, neighbor[0]) not in visited:
                    queue.append((neighbor[1], neighbor[0], node))  # Add neighbor with its weight and parent node
                    visited.add((neighbor[0], node))  # Mark edge as visited

    return bfs_order

# Graph representation: node -> list of (neighbor, weight)
Graph = {
    1: [[2, 2], [4, 1], [5, 4]],
    2: [[1, 2], [3, 3], [4, 3], [6, 7]],
    3: [[2, 3], [4, 5], [6, 8]],
    4: [[1, 1], [2, 3], [3, 5], [5, 9]],
    5: [[1, 4], [4, 9]],
    6: [[2, 7], [3, 8]],
}

# Perform BFS with weights
graph = bfsWithWeights(graph=Graph, start=1)
print("Graph return with BFS  -->  ", graph)
# Expected format of output: [(weight, node, parent), ...]
```

```python
# Initialize MST and visited nodes dictionary
mst = []
VISITED_NODE = {i: False for i in range(7)}

# Sort the edges by weight
graph.sort()
sum1 = 0  # Variable to keep track of the total weight of the MST

# Construct the MST
while len(graph) != 0:
    crrW, crrN1, crrN2 = graph.pop(0)  # Get the edge with the smallest weight
    if VISITED_NODE[crrN1] is False or VISITED_NODE[crrN2] is False:
        mst.append((crrN1, crrN2))  # Add edge to MST
        sum1 += crrW  # Add weight to total weight
        VISITED_NODE[crrN1] = True  # Mark nodes as visited
        VISITED_NODE[crrN2] = True

# Output the MST and its total weight
print("MST Edges:", mst)
print("Total weight of MST:", sum1)
```

Summary of the added comments:

1. **Initialization of BFS**:
    - Comments explaining the initialization of the queue, visited set, and BFS order list.
2. **BFS Loop**:
    - Comments explaining the dequeuing process and the condition for visiting nodes.
    - Comments explaining the enqueueing of adjacent nodes and marking edges as visited.
3. **Graph Representation**:
    - Comments describing the structure of the input graph.
4. **MST Initialization**:
    - Comments explaining the initialization of the MST list and the visited nodes dictionary.
5. **MST Construction Loop**:
    - Comments explaining the process of popping the smallest weight edge and adding it to the MST.
    - Comments describing the updating of the total weight and marking nodes as visited.
6. **Output**:
   - Comments describing the final output of the MST edges and its total weight.
   Sure, let's go through a dry run of the given algorithm step-by-step.

bfsWithWeights Function

1. **Initialization**:
   - `queue = deque([(0, start, -1)])` initializes the queue with the start node (1) with a weight of 0 and no parent (-1).
   - `visited = set()` initializes an empty set to keep track of visited edges.
   - `bfs_order = []` initializes an empty list to store the order of BFS traversal.
2. **First Iteration**:
   - Dequeue `(0, 1, -1)` from the queue.
   - Since node 1 is not in the `visited` set, it is marked as visited.
   - Node 1 has neighbors 2, 4, and 5.
   - Enqueue neighbors: `(2, 2, 1)`, `(1, 4, 1)`, and `(4, 5, 1)` into the queue.
   - Add edges `(2, 1)`, `(4, 1)`, and `(5, 1)` to the `visited` set.
3. **Second Iteration**:
   - Dequeue `(2, 2, 1)` from the queue.
   - Since node 2 is not in the `visited` set, it is marked as visited.
   - Node 2 has neighbors 1, 3, 4, and 6.
   - Enqueue neighbors: `(3, 3, 2)` and `(7, 6, 2)` into the queue.
   - Add edges `(3, 2)` and `(6, 2)` to the `visited` set.
4. **Third Iteration**:
   - Dequeue `(1, 4, 1)` from the queue.
   - Since node 4 is not in the `visited` set, it is marked as visited.
   - Node 4 has neighbors 1, 2, 3, and 5.
   - Enqueue neighbor: `(5, 3, 4)` into the queue.
   - Add edge `(5, 4)` to the `visited` set.
5. **Fourth Iteration**:
   - Dequeue `(4, 5, 1)` from the queue.
   - Since node 5 is not in the `visited` set, it is marked as visited.
   - Node 5 has neighbor 1 and 4.
6. **Fifth Iteration**:
   - Dequeue `(3, 3, 2)` from the queue.
   - Since node 3 is not in the `visited` set, it is marked as visited.
   - Node 3 has neighbors 2, 4, and 6.
   - Enqueue neighbor: `(8, 6, 3)` into the queue.
   - Add edge `(8, 3)` to the `visited` set.
7. **Sixth Iteration**:
   - Dequeue `(7, 6, 2)` from the queue.

- Since node 6 is not in the `visited` set, it is marked as visited.

Summary of BFS Traversal

- The `bfs_order` list will be `[(2, 2, 1), (1, 4, 1), (4, 5, 1), (3, 3, 2), (7, 6, 2)]`.

MST Construction

1. **Initialization**:
    - `mst = []` initializes an empty list to store the MST edges.
    - `VISITED_NODE` dictionary keeps track of visited nodes, all initialized to `False`.
2. **First Iteration**:
    - Pop `(1, 4, 1)` from the sorted `graph`.
    - Nodes 4 and 1 are not visited, so add edge `(4, 1)` to `mst` and mark nodes 4 and 1 as visited.
    - `mst = [(4, 1)]` and `sum1 = 1`.
3. **Second Iteration**:
    - Pop `(2, 2, 1)` from the sorted `graph`.
    - Nodes 2 and 1 are not visited, so add edge `(2, 1)` to `mst` and mark nodes 2 and 1 as visited.
    - `mst = [(4, 1), (2, 1)]` and `sum1 = 3`.
4. **Third Iteration**:
    - Pop `(3, 3, 2)` from the sorted `graph`.
    - Nodes 3 and 2 are not visited, so add edge `(3, 2)` to `mst` and mark nodes 3 and 2 as visited.
    - `mst = [(4, 1), (2, 1), (3, 2)]` and `sum1 = 6`.
5. **Fourth Iteration**:
    - Pop `(4, 5, 1)` from the sorted `graph`.
    - Nodes 5 and 1 are not visited, so add edge `(5, 1)` to `mst` and mark nodes 5 and 1 as visited.
    - `mst = [(4, 1), (2, 1), (3, 2), (5, 1)]` and `sum1 = 10`.
6. **Fifth Iteration**:
    - Pop `(7, 6, 2)` from the sorted `graph`.
    - Nodes 6 and 2 are not visited, so add edge `(6, 2)` to `mst` and mark nodes 6 and 2 as visited.
    - `mst = [(4, 1), (2, 1), (3, 2), (5, 1), (6, 2)]` and `sum1 = 17`.

Final Output

- The MST is `[(4, 1), (2, 1), (3, 2), (5, 1), (6, 2)]`.
- The total weight of the MST is `17`.