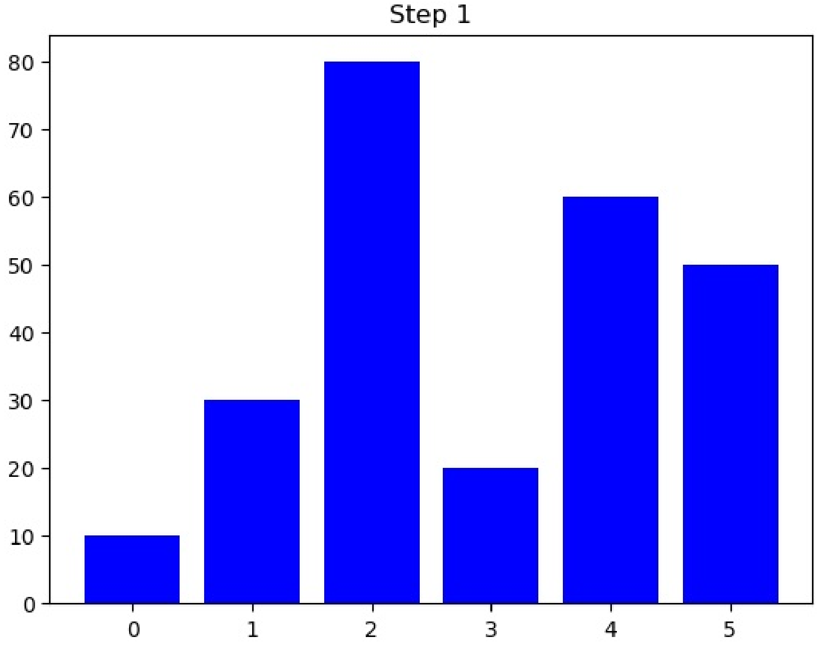
practical 3

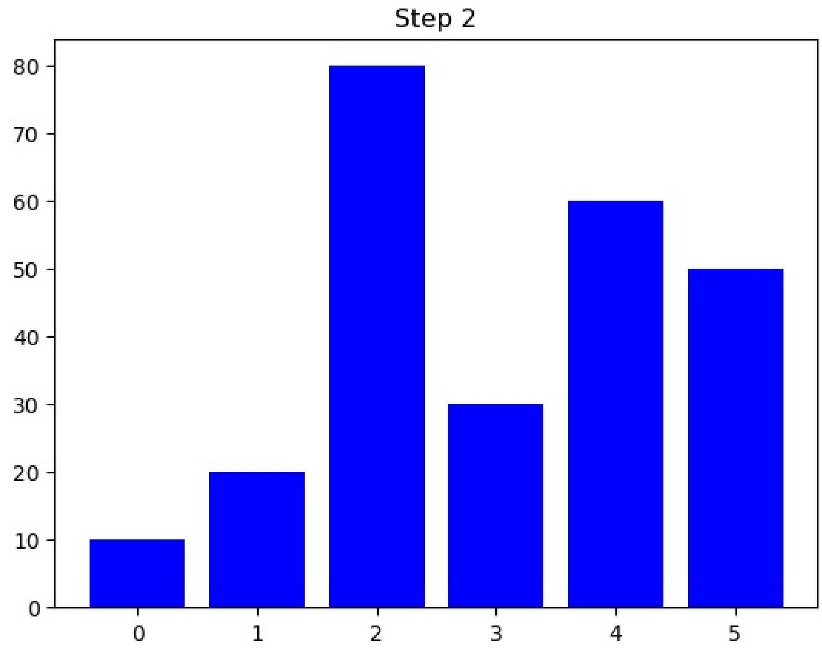
In [3]:

```
import matplotlib.pyplot as plt
import numpy as np
import time

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]  # Swap
        print(f"Pass {i + 1}: {arr}")  # Print pass output
        visualize_sort(arr, i)  # Visualization after each step

def visualize_sort(arr, step):
    plt.clf()
    plt.bar(range(len(arr)), arr, color='blue')
    plt.title(f"Step {step + 1}")
    plt.pause(0.5)

# User Input
arr = list(map(int, input("Enter numbers separated by space: ").split()))
print("Sorting steps:")
plt.ion()
selection_sort(arr)
plt.ioff()
plt.show()
print("Sorted array:", arr)
```

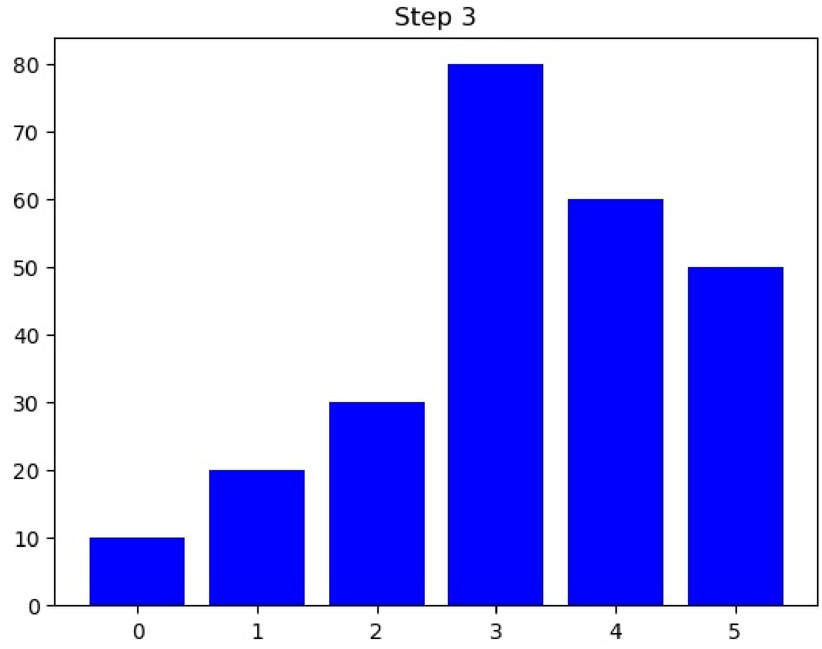
```
Enter numbers separated by space: 50 30 80 20 60 10
Sorting steps:
Pass 1: [10, 30, 80, 20, 60, 50]
```



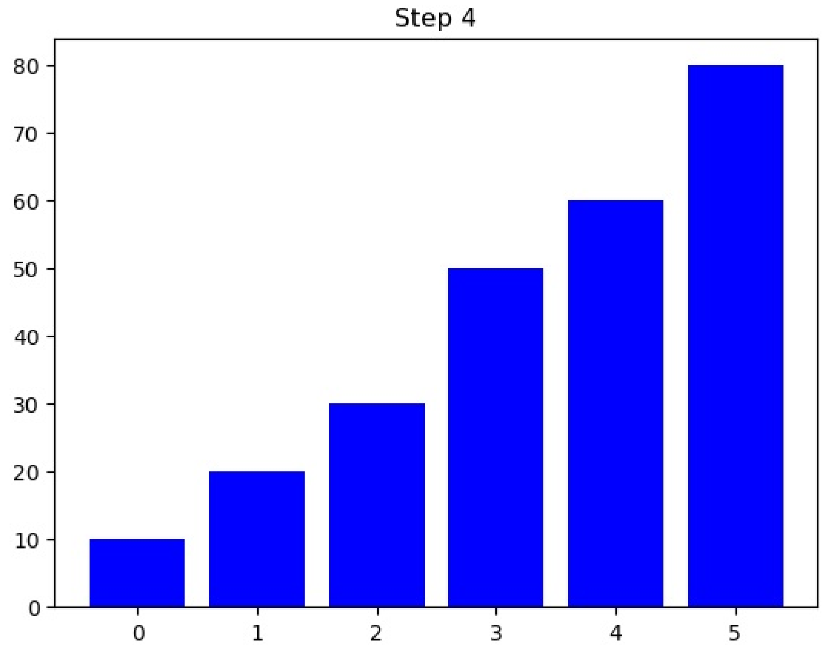
```
Pass 2: [10, 20, 80, 30, 60, 50]
```



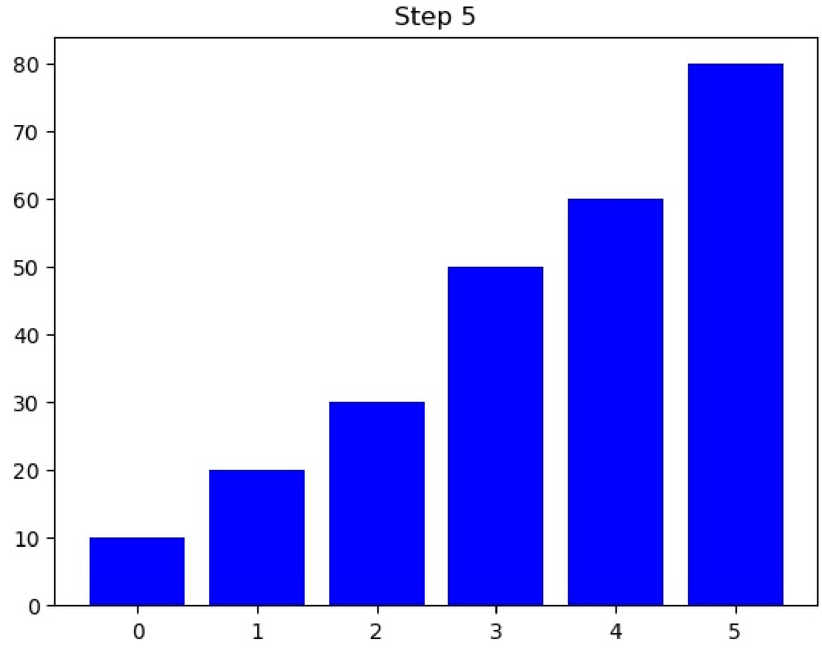
```
Pass 3: [10, 20, 30, 80, 60, 50]
```



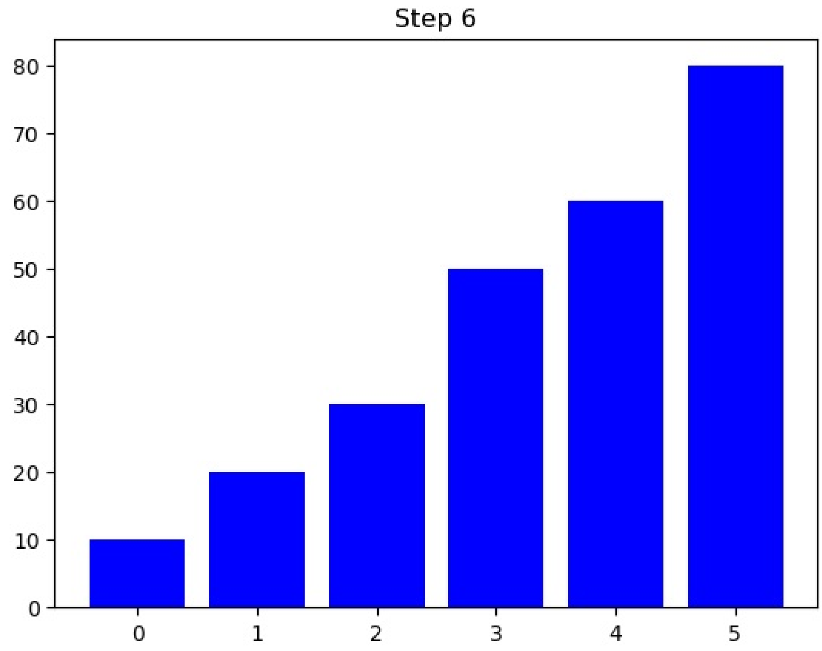
```
Pass 4: [10, 20, 30, 50, 60, 80]
```



```
Pass 5: [10, 20, 30, 50, 60, 80]
```



```
Pass 6: [10, 20, 30, 50, 60, 80]
```



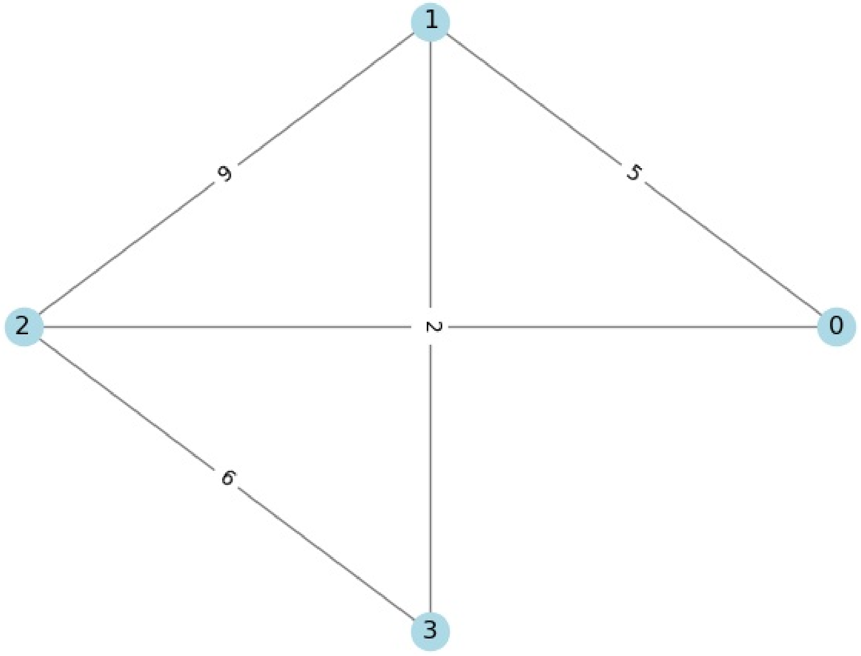
```
Sorted array: [10, 20, 30, 50, 60, 80]
```

In [9]:

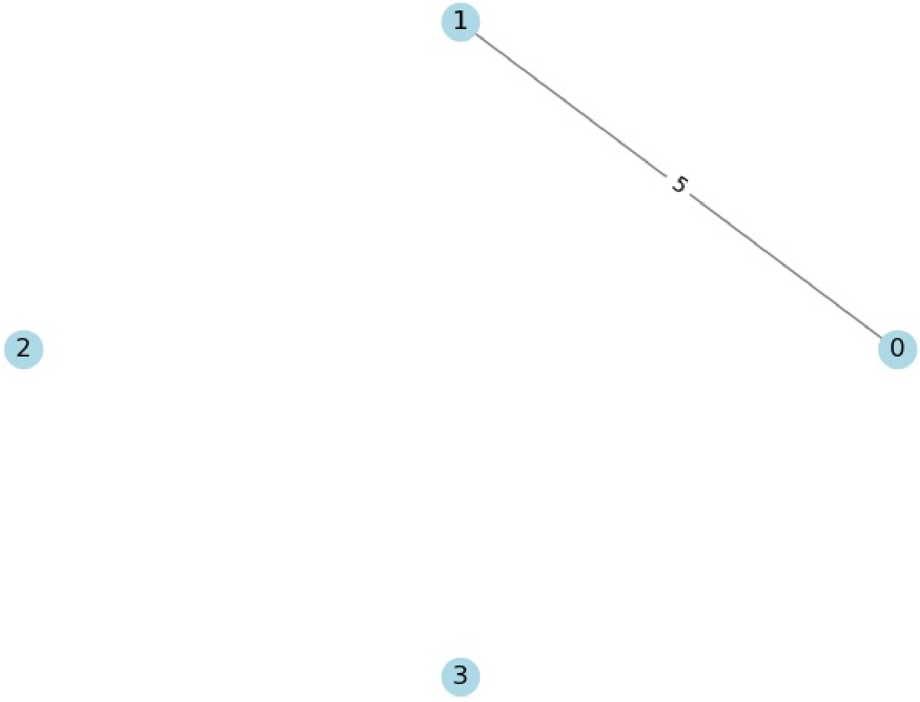
```python
import heapq
import networkx as nx
import matplotlib.pyplot as plt

def visualize_graph(graph, edges, title="Graph Visualization"):
    G = nx.Graph()
    for node in graph:
        G.add_node(node)
    for edge in edges:
        u, v, w = edge
        G.add_edge(u, v, weight=w)

    pos = nx.circular_layout(G)  # Arrange nodes in a circular layout
    labels = nx.get_edge_attributes(G, 'weight')

    plt.clf()
    nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray', edge_cmap=plt.cm.Blues)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
    plt.title(title)
    plt.pause(1)

def prims_algorithm(graph, start_node):
    n = len(graph)
    mst = []
    visited = set()
    min_heap = [(0, start_node, None)]  # (cost, node, parent)
    heapq.heapify(min_heap)
    total_cost = 0

    # Show the full initial graph before running Prim's Algorithm
    all_edges = []
    for node in graph:
        for neighbor, weight in graph[node]:
            if (neighbor, node, weight) not in all_edges:  # Avoid duplicate edges
                all_edges.append((node, neighbor, weight))
    plt.ion()
    visualize_graph(graph, all_edges, "Initial Graph")

    while len(mst) < n - 1 and min_heap:
        cost, node, parent = heapq.heappop(min_heap)

        if node in visited:
            continue

        visited.add(node)
        if parent is not None:
            mst.append((parent, node, cost))
            total_cost += cost
            visualize_graph(graph, mst, "Minimum Spanning Tree (Prim's Algorithm)")

        for neighbor, weight in graph[node]:
            if neighbor not in visited:
                heapq.heappush(min_heap, (weight, neighbor, node))

    print("Minimum Spanning Tree (MST) Edges:")
    for edge in mst:
        print(edge)
    print(f"Total Cost of MST: {total_cost}")

    plt.ioff()
    plt.show()
    return mst, total_cost

# User Input
graph = {}
n = int(input("Enter the number of nodes: "))

for _ in range(n):
    node = int(input(f"Enter node {_ + 1}: "))
    graph[node] = []
    edges_count = int(input(f"Enter the number of edges for node {node}: "))

    for _ in range(edges_count):
        neighbor, weight = map(int, input("Enter neighbor and weight (space-separated): ").split())
        graph[node].append((neighbor, weight))
        if neighbor not in graph:
            graph[neighbor] = []
        graph[neighbor].append((node, weight))

start_node = int(input("Enter start node: "))
prims_algorithm(graph, start_node)
```

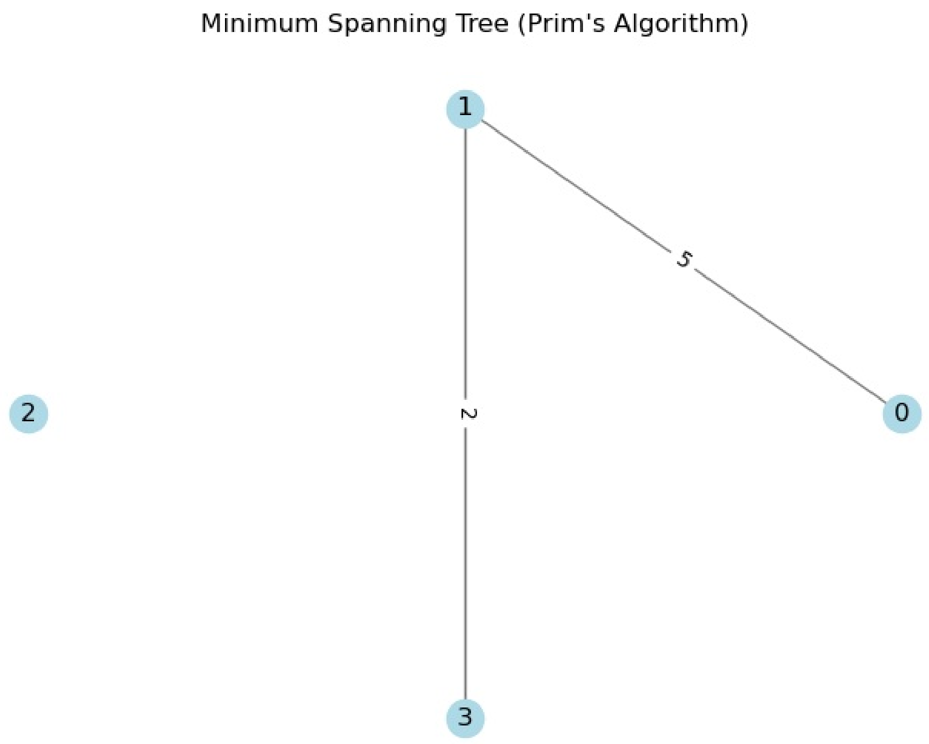
```
Enter the number of nodes: 4
Enter node 1: 0
Enter the number of edges for node 0: 2
Enter neighbor and weight (space-separated): 1 5
Enter neighbor and weight (space-separated): 2 8
Enter node 2: 1
Enter the number of edges for node 1: 3
Enter neighbor and weight (space-separated): 0 5
Enter neighbor and weight (space-separated): 2 9
Enter neighbor and weight (space-separated): 3 2
Enter node 3: 2
Enter the number of edges for node 2: 3
Enter neighbor and weight (space-separated): 1 9
Enter neighbor and weight (space-separated): 0 8
Enter neighbor and weight (space-separated): 3 6
Enter node 4: 3
Enter the number of edges for node 3: 2
Enter neighbor and weight (space-separated): 1 2
Enter neighbor and weight (space-separated): 2 6
Enter start node: 0
```

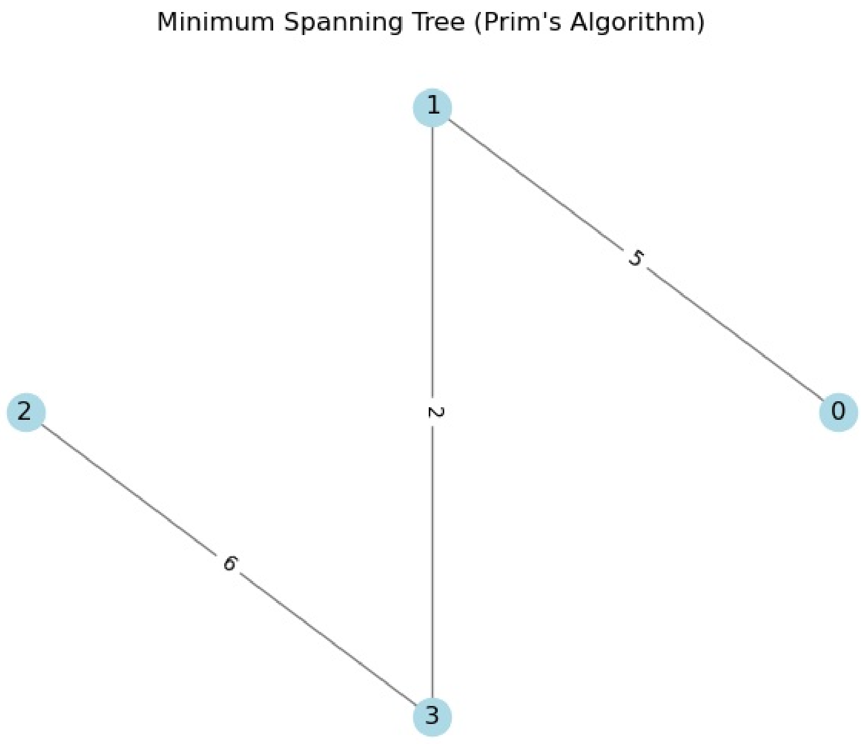
Initial Graph



Minimum Spanning Tree (Prim's Algorithm)







```
Minimum Spanning Tree (MST) Edges:
(0, 1, 5)
(1, 3, 2)
(3, 2, 6)
Total Cost of MST: 13
```

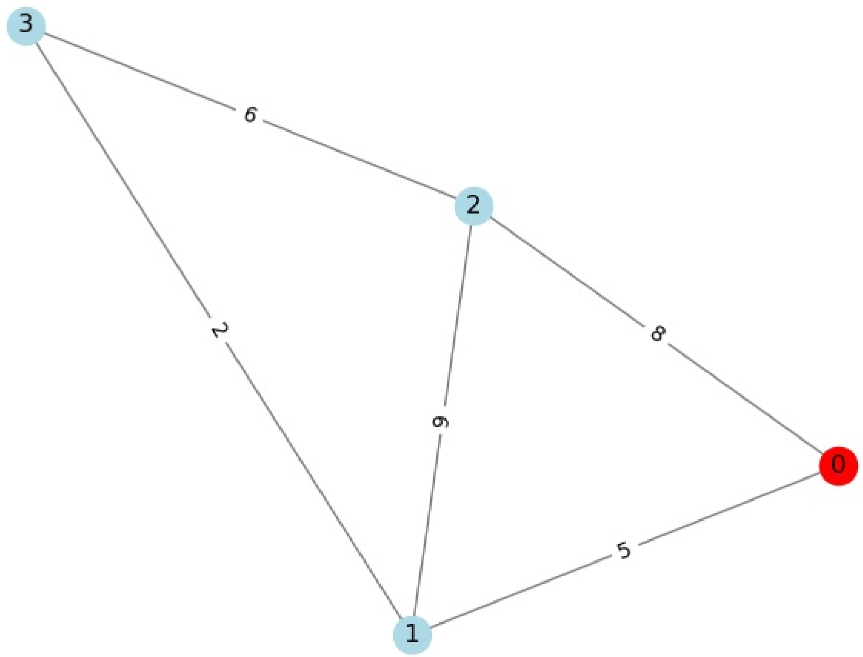
Out[9]:
```
([(0, 1, 5), (1, 3, 2), (3, 2, 6)], 13)
```

In [1]:
```
import heapq
import networkx as nx
import matplotlib.pyplot as plt

def visualize_dijkstra(graph, distances, visited):
    G = nx.Graph()

    for node in graph:
        G.add_node(node)

    for node, edges in graph.items():
        for neighbor, weight in edges:
            G.add_edge(node, neighbor, weight=weight)
```

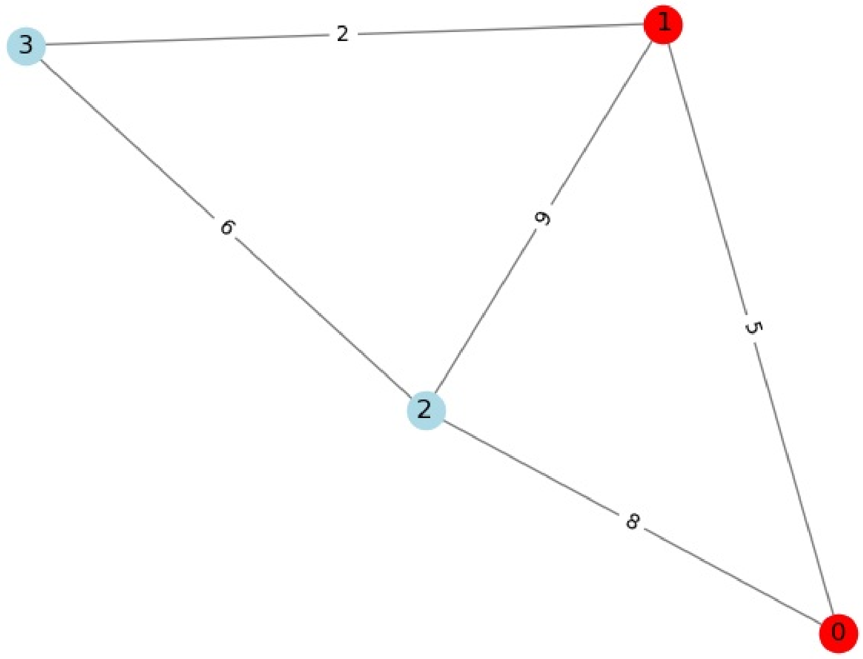
```python
    pos = nx.spring_layout(G)
    labels = nx.get_edge_attributes(G, 'weight')
    plt.clf()
    nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    # Highlight visited nodes
    nx.draw_networkx_nodes(G, pos, nodelist=visited, node_color='red')
    plt.title("Dijkstra's Algorithm Progress")
    plt.pause(1)

def dijkstra(graph, start):
    heap = [(0, start)]
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    visited = []

    while heap:
        cost, node = heapq.heappop(heap)
        if node in visited:
            continue

        visited.append(node)
        visualize_dijkstra(graph, distances, visited)  # Visualization

        for neighbor, weight in graph[node]:
            new_cost = cost + weight
            if new_cost < distances[neighbor]:
                distances[neighbor] = new_cost
                heapq.heappush(heap, (new_cost, neighbor))

    return distances

# Taking user input for graph construction
graph = {}
num_nodes = int(input("Enter the number of nodes: "))
for i in range(num_nodes):
    node = int(input(f"Enter node {i+1}: "))
    graph[node] = []
    num_edges = int(input(f"Enter the number of edges for node {node}: "))
    for _ in range(num_edges):
        neighbor, weight = map(int, input("Enter neighbor and weight (space-separated): ").split())
        graph[node].append((neighbor, weight))

start_node = int(input("Enter start node: "))
plt.ion()
print("Running Dijkstra's Algorithm...")
print("Shortest paths:", dijkstra(graph, start_node))
plt.ioff()
plt.show()
```

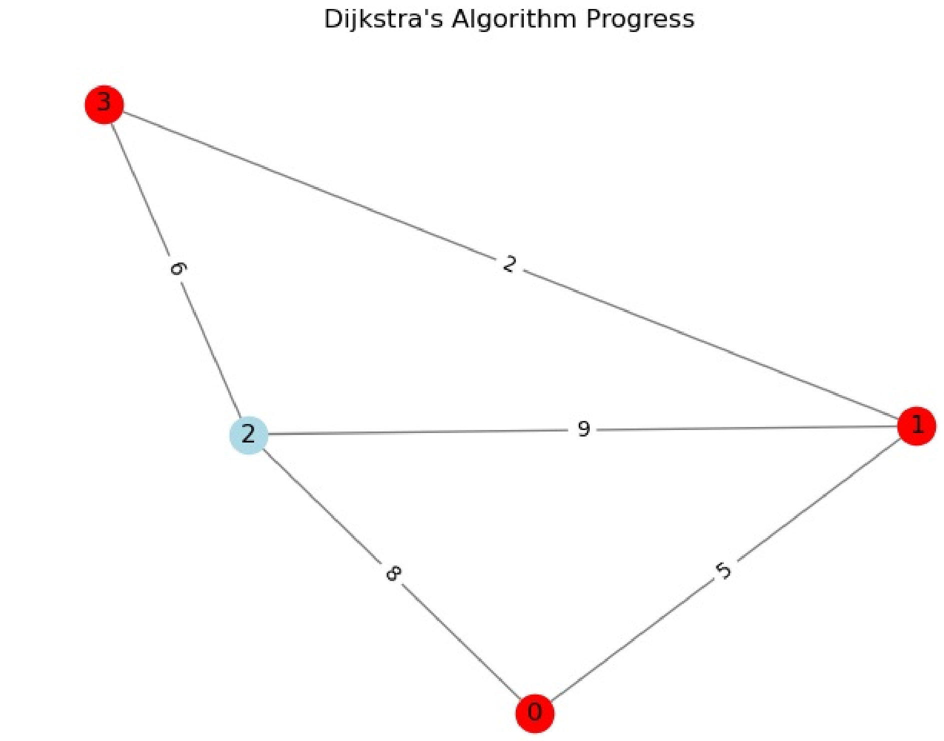
```
Enter the number of nodes: 4
Enter node 1: 0
Enter the number of edges for node 0: 2
Enter neighbor and weight (space-separated): 1 5
Enter neighbor and weight (space-separated): 2 8
Enter node 2: 1
Enter the number of edges for node 1: 3
Enter neighbor and weight (space-separated): 0 5
Enter neighbor and weight (space-separated): 2 9
Enter neighbor and weight (space-separated): 3 2
Enter node 3: 2
Enter the number of edges for node 2: 3
Enter neighbor and weight (space-separated): 0 8
Enter neighbor and weight (space-separated): 1 9
Enter neighbor and weight (space-separated): 3 6
Enter node 4: 3
Enter the number of edges for node 3: 2
Enter neighbor and weight (space-separated): 1 2
Enter neighbor and weight (space-separated): 2 6
Enter start node: 0
Running Dijkstra's Algorithm...
```

Dijkstra's Algorithm Progress

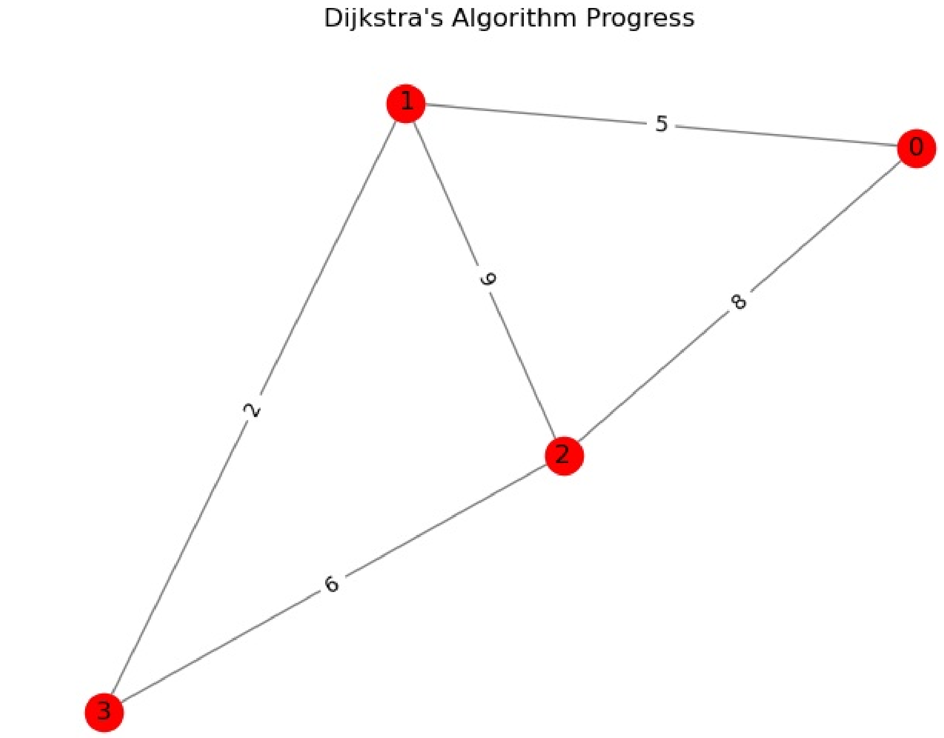


Dijkstra's Algorithm Progress





Dijkstra's Algorithm Progress



Dijkstra's Algorithm Progress

```
Shortest paths: {0: 0, 1: 5, 2: 8, 3: 7}
```

In [6]:

```python
def job_scheduling(jobs):
    jobs.sort(key=lambda x: x[1], reverse=True)
    result = []
    for job in jobs:
        result.append(job[0])
    return result

# User Input
n = int(input("Enter number of jobs: "))
jobs = []
for _ in range(n):
    name, profit = input("Enter job name and profit: ").split()
    jobs.append((name, int(profit)))

print("Scheduled Jobs:", job_scheduling(jobs))
```

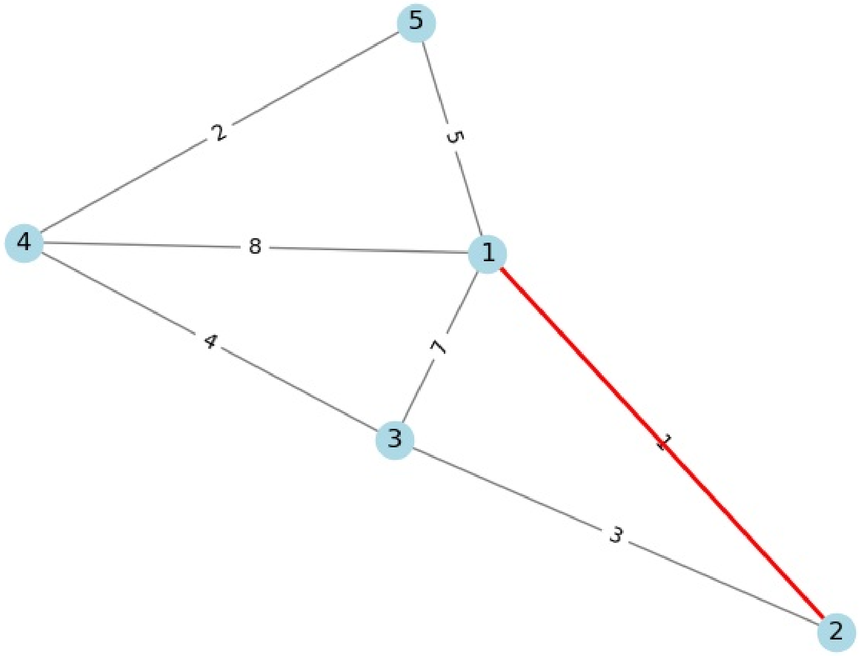
```
Enter number of jobs: 4
Enter job name and profit: a 12
Enter job name and profit: b 10
Enter job name and profit: c 14
Enter job name and profit: d 5
Scheduled Jobs: ['c', 'a', 'b', 'd']
```

In [5]:

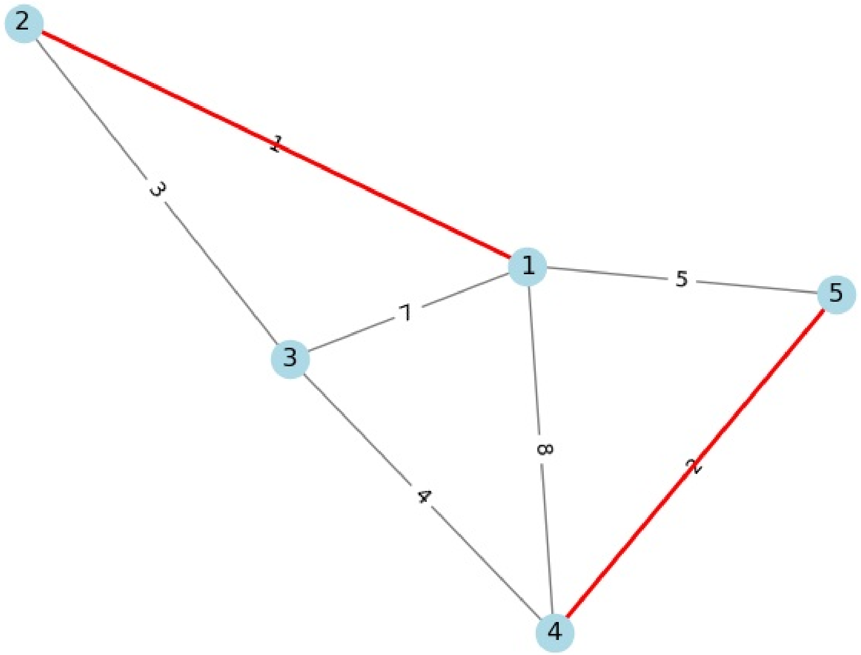
```python
import networkx as nx
import matplotlib.pyplot as plt

def visualize_kruskal(edges, mst):
    G = nx.Graph()
    for u, v, w in edges:
        G.add_edge(u, v, weight=w)

    pos = nx.spring_layout(G)
    labels = nx.get_edge_attributes(G, 'weight')
    plt.clf()
    nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    # Highlight MST edges
    mst_edges = [(u, v) for u, v, _ in mst]
    nx.draw_networkx_edges(G, pos, edgelist=mst_edges, edge_color='red', width=2)

    plt.title("Kruskal's MST Progress")
    plt.pause(1)

def kruskal(graph, n):
    edges = sorted(graph, key=lambda x: x[2])
    unique_nodes = set()
    for u, v, _ in edges:
        unique_nodes.add(u)
        unique_nodes.add(v)
    parent = {i: i for i in unique_nodes}

    def find(v):
        if parent[v] == v:
            return v
        parent[v] = find(parent[v])
        return parent[v]

    mst = []
    for u, v, weight in edges:
        pu, pv = find(u), find(v)
        if pu != pv:
            mst.append((u, v, weight))
            parent[pu] = pv
            visualize_kruskal(edges, mst)  # Visualization

    return mst

# User Input
edges = []
n = int(input("Enter number of nodes: "))
for _ in range(int(input("Enter number of edges: "))):
    u, v, w = map(int, input("Enter edge (u, v, weight): ").split())
    edges.append((u, v, w))

plt.ion()
print("Running Kruskal's Algorithm...")
kruskal(edges, n)
plt.ioff()
plt.show()
```

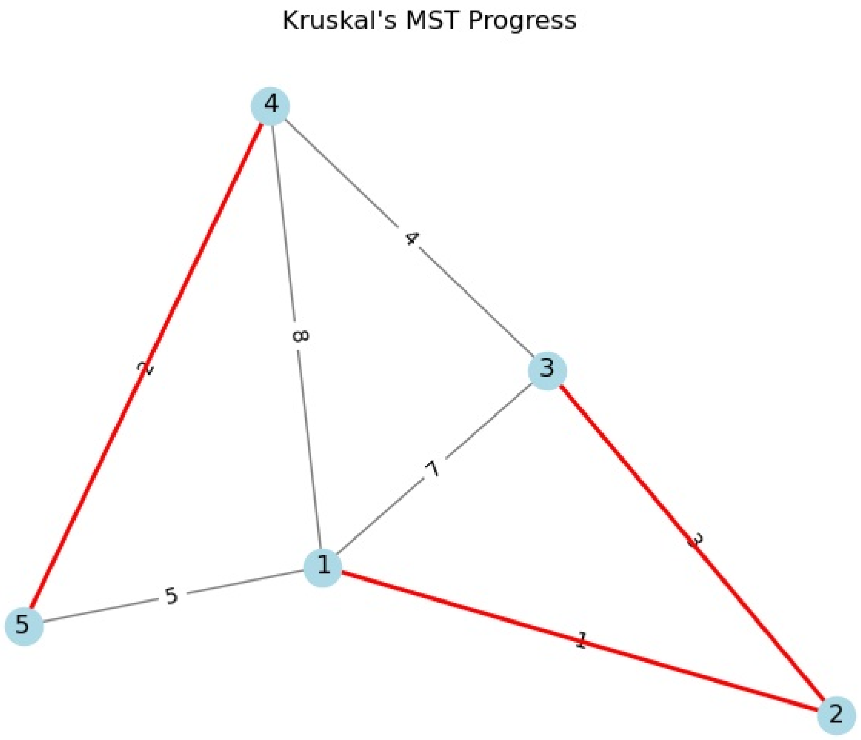
```
Enter number of nodes: 5
Enter number of edges: 7
Enter edge (u, v, weight): 1 2 1
Enter edge (u, v, weight): 2 3 3
Enter edge (u, v, weight): 3 4 4
Enter edge (u, v, weight): 4 5 2
Enter edge (u, v, weight): 5 1 5
Enter edge (u, v, weight): 1 4 8
Enter edge (u, v, weight): 1 3 7
Running Kruskal's Algorithm...
```

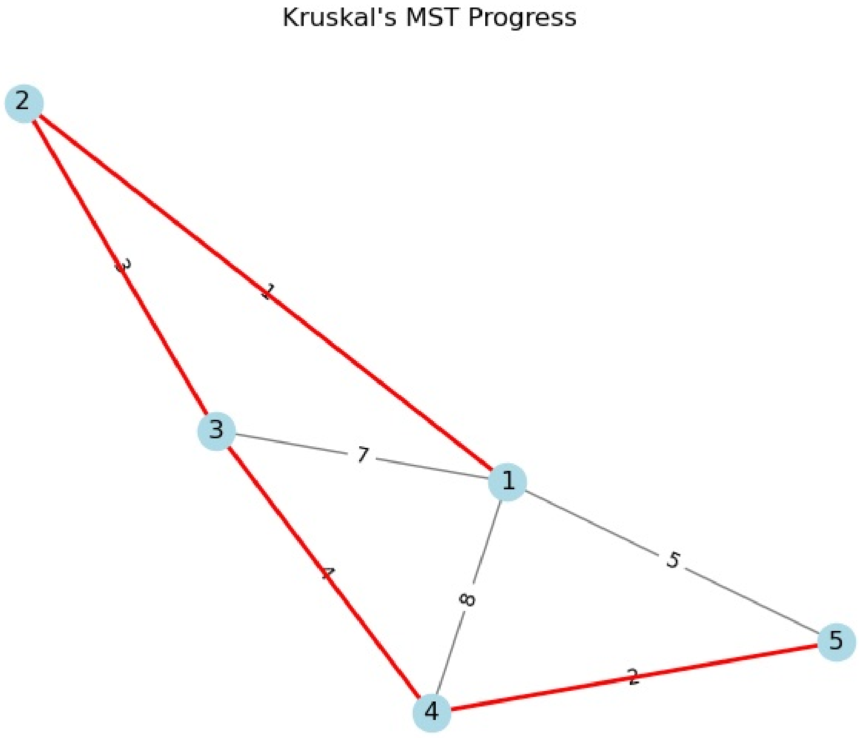
Kruskal's MST Progress



Kruskal's MST Progress




Kruskal's MST Progress
4
8
4
2
3
7
3
1
5
1
5
2


Kruskal's MST Progress
2
3
1
3
7
1
5
4
8
5
2
4

#

In [ ]:

In [ ]: