In [2]:
```
#Selection Sort
def selection_sort(arr):

    for i in range(len(arr)):

         min_index = i
    for j in range(i+1, len(arr)):
        if arr[j] < arr[min_index]:
             min_index = j
    arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr
a1=[20,10,5,7,9,13]
selection_sort(a1)
```

Out[2]:
```
[20, 10, 5, 7, 9, 13]
```

In [5]:
```
#Prims Algorithm
import heapq

def prim(graph, start):
    mst = []
    visited = set([start])
    edges = [ (cost, start, to) for to, cost in graph[start].items() ]
    heapq.heapify(edges)

    while edges:
        cost, frm, to = heapq.heappop(edges)
        if to not in visited:
            visited.add(to)
            mst.append((frm, to, cost))

            for to_next, cost2 in graph[to].items():
                 if to_next not in visited:
                     heapq.heappush(edges, (cost2, to, to_next))

    return mst
```

In [7]:
```
graph = {
 'A': {'B': 2, 'C': 3},
 'B': {'A': 2, 'C': 1, 'D': 1},
 'C': {'A': 3, 'B': 1, 'D': 4},
 'D': {'B': 1, 'C': 4},
}
print(prim(graph,'A'))
```

```
[('A', 'B', 2), ('B', 'C', 1), ('B', 'D', 1)]
```

In [8]:
```python
#Kruskals Algorithm
def kruskal(graph):

    mst = []
    edges = [(cost, frm, to) for frm, to_dict in graph.items() for to, cost in
    edges.sort()
    parent = {node: node for node in graph}

    def find_root(node):
        if parent[node] == node:
            return node
        parent[node] = find_root(parent[node])
        return parent[node]

    for cost, frm, to in edges:
        root1 = find_root(frm)
        root2 = find_root(to)
        if root1 != root2:
            parent[root1] = root2
            mst.append((frm, to, cost))
    return mst
```

In [10]:
```python
graph = {
 'A': {'B': 2, 'C': 3},
 'B': {'A': 2, 'C': 1, 'D': 1},
 'C': {'A': 3, 'B': 1, 'D': 4},
 'D': {'B': 1, 'C': 4},
}
print(kruskal(graph))
```

```
[('B', 'C', 1), ('B', 'D', 1), ('A', 'B', 2)]
```

In [ ]: