1 .SELECTION SORT

```
def selectionSort(arr):
    n = len(arr)
    for i in range(n-1):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[min_idx], arr[i] = arr[i], arr[min_idx]
n = int(input("Enter the size of the input: "))
arr = []
print("Enter the elements of the array:")
for i in range(n):
    element = int(input("Enter the {} element: ".format(i+1)))
    arr.append(element)
print ("Unsorted array:")
for num in arr:
    print(num, end=" ")
selectionSort(arr)
print("\nSorted array:")
for num in arr:
    print(num, end=" ")
```

OUTPUT :

```
Enter the size of the input: 5
Enter the 1 element: 64
Enter the 2 element: 25
Enter the 3 element: 12
Enter the 4 element: 23
Enter the 5 element: 16
Unsorted array: 64 25 12 23 16
Sorted array:12 16 23 25 64
```

2. DIJIKTRAS MST ALGORITHM

```
import sys
class DijkstraMST:
    def __init__(self, graph, num_vertices):
        self.num_vertices = num_vertices
        self.graph = graph
        self.dist = [sys.maxsize] * num_vertices
        self.visited = [False] * num_vertices
    def dijkstra(self, start_vertex):
        for i in range(self.num_vertices):
            self.dist[i] = sys.maxsize
            self.visited[i] = False
        self.dist[start_vertex] = 0
        for _ in range(self.num_vertices - 1):
            u = self.min_distance(self.dist, self.visited)
            self.visited[u] = True
            for v in range(self.num_vertices):
                if not self.visited[v] and self.graph[u][v] != 0 and self.dist[u] != sys.maxsize and self.dist[u] +
self.graph[u][v] < self.dist[v]:
                    self.dist[v] = self.dist[u] + self.graph[u][v]
        self.print_mst(start_vertex)
    def min_distance(dist, visited):
        min_dist = sys.maxsize
        min_index = -1
        for i in range(len(dist)):
            if not visited[i] and dist[i] <= min_dist:
                min_dist = dist[i]
                min_index = i
        return min_index
```

```
    def print_mst(self, start_vertex):
        print("Vertex \t Distance from Source")
        for i in range(self.num_vertices):
            print(i, "\t", self.dist[i])
if __name__ == "__main__":
    n = int(input("Enter the size of the graph: "))
    graph = []
    for _ in range(n):
        row = []
        for j in range(n):
            weight = int(input("Enter the weight {}->{} of the graph: ".format(_, j)))
            row.append(weight)
        graph.append(row)
    start_vertex = int(input("Enter the starting vertex of the graph: "))
    dijkstra = DijkstraMST(graph, n)
    dijkstra.dijkstra(start_vertex)
```

OUTPUT :

```
Enter the  size of the graph: 5
Enter the weight 0-> 0 of the graph: 0
Enter the weight 0-> 1 of the graph: 2
Enter the weight 0-> 2 of the graph: 0
Enter the weight 0-> 3 of the graph: 6
Enter the weight 0-> 4 of the graph: 0
Enter the weight 1-> 0 of the graph: 2
Enter the weight 1-> 1 of the graph: 0
Enter the weight 1-> 2 of the graph: 3
Enter the weight 1-> 3 of the graph: 8
Enter the weight 1-> 4 of the graph: 5
Enter the weight 2-> 0 of the graph: 0
```

Enter the weight 2-> 1 of the graph: 3

Enter the weight 2-> 2 of the graph: 0

Enter the weight 2-> 3 of the graph: 0

Enter the weight 2-> 4 of the graph: 7

Enter the weight 3-> 0 of the graph: 6

Enter the weight 3-> 1 of the graph: 8

Enter the weight 3-> 2 of the graph: 0

Enter the weight 3-> 3 of the graph: 0

Enter the weight 3-> 4 of the graph: 9

Enter the weight 4-> 0 of the graph: 0

Enter the weight 4-> 1 of the graph: 5

Enter the weight 4-> 2 of the graph: 7

Enter the weight 4-> 3 of the graph: 9

Enter the weight 4-> 4 of the graph: 0

Enter the starting vertex of the graph: 1

Vertex Distance from Source

0 2

1 0

2 3

3 8

4 5

3. PRIMS MST ;

```
def minKey(key, mstSet):
    min = float('inf')
    minIndex = -1
    for i in range(len(key)):
        if not mstSet[i] and key[i] < min:
            min = key[i]
            minIndex = i
    return minIndex
```

```
def printMST(parent, graph, sum):
    print("Edge \tWeight")
    for i in range(1, len(parent)):
        print(parent[i], "-", i, "\t", graph[i][parent[i]])
    print("Minimum weight of MST:", sum)
def prim(graph, numVertices):
    parent = [0] * numVertices
    key = [float('inf')] * numVertices
    mstSet = [False] * numVertices
    key[0] = 0
    parent[0] = -1
    for count in range(numVertices - 1):
        u = minKey(key, mstSet)
        mstSet[u] = True
        for v in range(numVertices):
            if graph[u][v] != 0 and not mstSet[v] and graph[u][v] < key[v]:
                parent[v] = u
                key[v] = graph[u][v]
    sum = 0
    for i in range(numVertices):
        sum += key[i]
    printMST(parent, graph, sum)
n = int(input("Enter the size of the graph: "))
graph = [[0] * n for _ in range(n)]
for i in range(n):
    for j in range(n):
        graph[i][j] = int(input("Enter the weight {}->{} of the graph: ".format(i, j)))
prim(graph, n)
```

OUTPUT:

```
Enter the  size of the graph: 5
```

Enter the weight 0-> 0 of the graph: 0

Enter the weight 0-> 1 of the graph: 2

Enter the weight 0-> 2 of the graph: 0

Enter the weight 0-> 3 of the graph: 6

Enter the weight 0-> 4 of the graph: 0

Enter the weight 1-> 0 of the graph: 2

Enter the weight 1-> 1 of the graph: 0

Enter the weight 1-> 2 of the graph: 3

Enter the weight 1-> 3 of the graph: 8

Enter the weight 1-> 4 of the graph: 5

Enter the weight 2-> 0 of the graph: 0

Enter the weight 2-> 1 of the graph: 3

Enter the weight 2-> 2 of the graph: 0

Enter the weight 2-> 3 of the graph: 0

Enter the weight 2-> 4 of the graph: 7

Enter the weight 3-> 0 of the graph: 6

Enter the weight 3-> 1 of the graph: 8

Enter the weight 3-> 2 of the graph: 0

Enter the weight 3-> 3 of the graph: 0

Enter the weight 3-> 4 of the graph: 9

Enter the weight 4-> 0 of the graph: 0

Enter the weight 4-> 1 of the graph: 5

Enter the weight 4-> 2 of the graph: 7

Enter the weight 4-> 3 of the graph: 9

Enter the weight 4-> 4 of the graph: 0

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

Minimum weight of MST: 16

4. KRUSKAL MST :

```
class Edge:
    def __init__(self, src, dest, weight):
        self.src = src
        self.dest = dest
        self.weight = weight
    def __lt__(self, other):
        return self.weight < other.weight
def find(parent, i):
    if parent[i] != i:
        parent[i] = find(parent, parent[i])
    return parent[i]
def kruskal(graph, numVertices):
    edges = []
    for i in range(numVertices):
        for j in range(i + 1, numVertices):
            if graph[i][j] != 0:
                edge = Edge(i, j, graph[i][j])
                edges.append(edge)
    edges.sort()
    parent = list(range(numVertices))
    mst = []
    total_weight = 0
    for edge in edges:
        src_parent = find(parent, edge.src)
        dest_parent = find(parent, edge.dest)
        if src_parent != dest_parent:
            mst.append(edge)
            parent[src_parent] = dest_parent
            total_weight += edge.weight
```

```
    print("Edges in the MST:")
    for edge in mst:
        print(edge.src, "-", edge.dest, ":", edge.weight)
    print("Minimum weight of MST:", total_weight)
n = int(input("Enter the size of the graph: "))
graph = []
for i in range(n):
    row = []
    for j in range(n):
        weight = int(input("Enter the weight " + str(i) + "->" + str(j) + " of the graph: "))
        row.append(weight)
    graph.append(row)
kruskal(graph, n)
```

OUTPUT:

```
Enter the  size of the graph: 5
Enter the weight 0-> 0 of the graph: 0
Enter the weight 0-> 1 of the graph: 2
Enter the weight 0-> 2 of the graph: 0
Enter the weight 0-> 3 of the graph: 6
Enter the weight 0-> 4 of the graph: 0
Enter the weight 1-> 0 of the graph: 2
Enter the weight 1-> 1 of the graph: 0
Enter the weight 1-> 2 of the graph: 3
Enter the weight 1-> 3 of the graph: 8
Enter the weight 1-> 4 of the graph: 5
Enter the weight 2-> 0 of the graph: 0
Enter the weight 2-> 1 of the graph: 3
Enter the weight 2-> 2 of the graph: 0
Enter the weight 2-> 3 of the graph: 0
Enter the weight 2-> 4 of the graph: 7
```

```
Enter the weight 3-> 0 of the graph: 6
Enter the weight 3-> 1 of the graph: 8
Enter the weight 3-> 2 of the graph: 0
Enter the weight 3-> 3 of the graph: 0
Enter the weight 3-> 4 of the graph: 9
Enter the weight 4-> 0 of the graph: 0
Enter the weight 4-> 1 of the graph: 5
Enter the weight 4-> 2 of the graph: 7
Enter the weight 4-> 3 of the graph: 9
Enter the weight 4-> 4 of the graph: 0
Edge     Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Minimum weight of MST: 16
```

5. JOB SHEDULING :

```
class Job:
    def __init__(self, id, deadline, profit):
        self.id = id
        self.deadline = deadline
        self.profit = profit
n = int(input("Enter the no of Job you want to enter: "))
jobs = []
print("Enter the details of the Job:")
for i in range(n):
    print("Job", i + 1, ":")
    id = int(input("Enter the id of Job: "))
    deadline = int(input("Enter the deadline of Job: "))
    profit = int(input("Enter the profit of Job: "))
```

```
    jobs.append(Job(id, deadline, profit))
jobs.sort(key=lambda x: x.profit, reverse=True)
maxDeadline = max(job.deadline for job in jobs)
slots = [0] * (maxDeadline + 1)
totalProfit = 0
for job in jobs:
    for i in range(job.deadline, 0, -1):
        if slots[i] == 0:
            slots[i] = job.id
            totalProfit += job.profit
            break
print("Scheduled Jobs:", end=" ")
for i in range(1, len(slots)):
    if slots[i] != 0:
        print(slots[i], end=" ")
print("\nTotal Profit:", totalProfit)
```

OUTPUT

```
Enter the no of Job you want to enter: 5
Enter the details of the Job:
Job 1 :
Enter the id of Job: 1
Enter the deadline of Job: 2
Enter the profit of Job: 100
Job 2 :
Enter the id of Job: 2
Enter the deadline of Job: 1
Enter the profit of Job: 19
Job 3 :
Enter the id of Job: 3
```

Enter the deadline of Job: 2

Enter the profit of Job: 27

Job 4 :

Enter the id of Job: 4

Enter the deadline of Job: 1

Enter the profit of Job: 25

Job 5 :

Enter the id of Job: 5

Enter the deadline of Job: 3

Enter the profit of Job: 15

Scheduled Jobs: 3 1 5

Total Profit: 142