20101537

Ans. to the question no. 03:

```
function Dijkstra(Graph, source):
    dist[source] ← 0
    queue = PriorityQueue()
    visited = [None] * (length(Graph) + 1)
    prev = [None] * (length(Graph) + 1)
    for each vertex V in graph : ⟶ O(V)
        if V ≠ source :
            dist[v] = ∞
            prev[v] = null
    queue.put((dist[source], source))
    while queue not empty : ⟶ O(E log
        U = queue.extract_min() ⟶ log E
        if visited[U]:
            continue
        visited[U] ← True
        for each neighbour v of U: ⟶ O(E)
            alt = dist[U] + length(U, V)
            if alt < dist[v]:
                dist[v] = alt
                prev[v] = U
                queue.put((distance[v], U)) ⟶ O(log E)
```

For task 1, if we use adjacency list then the time complexity will be $(O(V+E)\log(V))$. If we use adjacency matrix then the time complexity will be $O(V^2 + E\log(V))$.

For task2, same as task 1 pseudo code # just adding the path-

```
flag = end_point
temp = [ ]
temp.append(flag)
temp.
while flag != source ⟶ O(V)
```

as adding $O(V)$ with the time complexity of the Dijikstra pseudo code will not add any significance to the time complexity. so we ignore it.

Time complexity for task 2.

For adjacency list - $O((V+E)\log(V))$

" " Matrix = $O(V^2 + E\log(V))$

PART 2

If the number of titans in each road is exactly 1 or weight of each edge is 1 we can use BFS Algorithm to solve the problem. The time complexity of BFS is $O(N+M)$.

for adjacency matrix it will be $O(N^2)$

Algo →

```
function BFS (visited, graph, node, endpoint):
        visited [node] = 1
        queue.put (node)
O(N) ← while queue not empty:
            m = queue.get()
            if m = endpoint:
                return
O(M) ← for each neighbour of m in graph:
                if visited [neighbour] = 0
                visited [neighbour] = 1
                queue.put (neighbour)
```

Ans. to the question no. 04:

We know BFS also gives shortest path between source and destination But in this case each edge have weights of each route / path. We know we can't use BFS for weighted graph to find shortest path. So we used Dijkstra to find a path from Motijheel to Moghbazar.