Task-3

In task-1:

```
def Dijkstra(dict1, dict2, source):
    wei = [float('inf')]*(len(dict2)+1))]
    par = [None]*(len(dict2+1)
    prior_q = {}
    visi = [False]*(len(dict2)+1)'

    wei[source] = 0

    prior_q[wei[source]] = source

    while prior_q != {}:
        m = min(prior_q.keys())
        min1 = prior_q.pop(m)

        if visi[min1]:
            continue

    visi[min1] = True
```

O(V) +
O(logV)
→ O(V)

```
for nei in dict2[min1]:
    check = wei[min1] + dict1[min1, nei]

    if check < wei[nei] ]:                  → O(ElogV)
        wei[nei] = check
        par[nei] = min1

        prior_q[wei[nei]] = nei
```

∴ Time complexity of of this algorithm is O(MlogN) and where for the adjacency list it is O(M+N)

∴ Total time of complexity = O(MlogN) + O(M+N)

In task-2

```
def Dijkstra (dict1, dict2, source):
    wei = [float('inf')] * (len(dict2)+1)
    par = [None] * (len(dict2+1)
    priorq = {}
    visi = [False] * (len(dict2)+1))

   wei[source] = 0

  priorq[wei[source]] = source

  while prior-q != {}:
        m = min(prior-q.keys())
        min1 = prior-q.pop(m)

       if visit[min1]:
            continue
       visi[min1] = True
```

O(V) + O(log(V)) = O(V)

```
for nei in dict2[min1]:
    check = wei[min1] + dict1[min1, nei]
    if check < wei[nei]:                              O(Elog V)
        wei[nei] = check
        par[nei] = min1

        prior_ar[wei[nei]] = nei

path = []

x = len (par) - 1
path.append(x)

while par[x] != None:
    path.append(par[x])                               O(V)
    x = par[x]

path.reverse()
```

So, here for adjacency list it is $O(M+N)$
and for the algorithm it is $O(M \log N)$

$\therefore$ Time complexity $= O(M+N) + O(M \log N)$

If the number of titans in each road is exactly 1.. then the algorithm we can use is Breadth First Search which is BFS.