Qn – 3:

Design an algorithm that given a pair of cities a, b, computes the cheapest possible route (i.e. sequence of flights) that starts at a at time 0, and ends at b at time at most T. The running time of your algorithm should be polynomial in n and m (e.g. O(n10m5)).

There are n cities and m flights connecting the cities
Let X1, X2, … Xm denote list of starting point of m flights and Let Y1, Y2, … Ym denote list of destination point of m flights
Let DT1, DT2, … DTm denote list of departure time of m flights
Let DU1, DU2, … DUm denote list of duration of m flights
Let CO1, CO2, … COm denote list of cost of m flights
Let FL be list of all explored flights and Cl be list of all explored Cities
Let List - {Ti(City), C(City)} be list of all possible times after which you can board a flight from a city (i.e. list of all possible times you can arrive at a city) and cost of reaching the city in time Ti. Initially this list is empty for each city

Algorithm:

Consider each city as a node in a directed graph and each flight as directed edge connecting two cities

- **To reach b from a with the cheapest possible route, so that you start from a at time 0 and reach b at time at most T, explore one flight at a time in the order of cheapest flights first out of those which satisfy the constraints (for time T and departure time availability)**
- **Stop algorithm when a flight which satisfies the constraints and with destination as city b is explored**
- **If all possible flights are explored which don't violate the constraints and node b is not reached then there is no possible route from a to b which satisfies the constraints**

Cl = {a}, { Ti(a), C(a) } = {0, 0}

While at least one flight is still unexplored:

Choose list of flights f not explored with starting point u in Cl and any destination v(may or may not be in Cl) which satisfy the following constraints:

1. {DT(f) – Starting Time(a)} + DU(f) < T
2. There exists a path from source city a to u, such that you can arrive at u before departure of flight f (i.e.)
   DT(f) >= Ti(u)

If no flights satisfy these constraints:

Msg – **"No possible route from a to b within time T"**
Break loop

Out of the list of flights that satisfy the constraints, chose the flight with the smallest value of the following

Min(Cost of reaching u from a which satisfies the constraints) + CO(f)

If v already present in Cl

Add f as another possible path to reach v
Add {DT(f) + DU(f), Min(Cost of reaching u from a which satisfies constraints) + CO(f)} to List of {Ti(v), C(v)}
Mark f as explored flight

If v not in Cl

Add v to Cl

Add f as another possible path to reach v

Add {DT(f) + DU(f), Min(Cost of reaching u from a which satisfies constraints) + CO(f)} to List of {Ti(v), C(v)}

Mark f as explored flight

If v == b (destination city)

Return (cheapest route from a->u which satisfy the constraints) + f as cheapest route to b from a

Proof of Termination:

The algorithm explores at least one flight for iteration and in worst case scenario it runs for m flights and since no already explored flight is explored again it terminates after m iterations

Proof of correctness:

- This algorithm is similar to the Dijkstra's algorithm with the difference being that even if a city is explored via some flight we continue to explore other flights to v
- This is because as the current path to v might be cheaper but might reach later than the corresponding flight leaving from v
- Hence in this algorithm each flight which satisfies the constraints is explored in the increasing order of cost from source city a
- Therefore the first flight we explore which reaches the destination city b will be the cheapest possible route available to reach city B

Time Complexity:

- The algorithm runs in the worst case for m iterations (number of flights)
- For each iteration, we check all unexplored flights (their departure times and duration) and check the corresponding departure cities of the flights if the arrival time at the city is before the departure of the flight
- Hence each iteration has a complexity of O(nm)
- Totally the algorithm has a complexity of **$O(nm^2)$**